Volume

1

UNIFIED PROCESS & UNIFIED MODELING LANGUAGE

LABORATÓRIO DE ENGENHARIA DE SOFTWARE

Gerenciamento de Requisitos de Software

LABORATÓRIO DE ENGENHARIA DE SOFTWARE

Gerenciamento de Requisitos de Software

Leffingwell, Dean & Widrig, Don. Managing Software Requirements: A Unified Approach – Addison-Wesley object technology series, Addison Wesley, 2000. ISBN: 0-201-61593-2

Tradução e Revisão Técnica

© Osvaldo Kotaro Takai,
e-mail: otakai@uol.com.br

Índice

ÍNDICE **3**

CAPÍTULO 1 ..... **8**

O Problema da Pedra (por Ed Yourdon) 8

CAPÍTULO 2 ..... **11**

Introdução ao Gerenciamento de Requisitos 11

Definições *11*

O que é um Requisito? 11

O que é o Gerenciamento de Requisitos? 11

Aplicação das Técnicas de Gerenciamento de Requisitos *13*

Tipos de Aplicações de Software 13

Aplicações de Sistemas 13

O Mapa da Mina *14*

O Domínio do Problema 14

Necessidades dos Stakeholders 15

Caminhando em Direção ao Domínio da Solução 15

Características do Sistema 15

Requisitos de Software 15

Uma Introdução aos Use Cases 16

Resumo *16*

CAPÍTULO 3 ..... **17**

A Equipe de Software 17

Desenvolvimento de Software como uma Atividade de Equipe *18*

Habilidades da Equipe de Requisitos para o Gerenciamento Efetivo de Requisitos 18

Membros da Equipe possuem Habilidades Distintas 19

A Organização da Equipe de Software 19

O Caso de Estudo *20*

Escopo do Caso de Estudo 20

A Equipe de Desenvolvimento do Software HOLIS 21

Sumário *22*

HABILIDADE DE EQUIPE 1 ..... **23**

Analisando o Problema 23

CAPÍTULO 4 ..... **25**

Os Cinco Passos da Análise do Problema 25

Passo 1: Chegar ao Acordo sobre a Definição do Problema *26*

A Declaração do Problema 27

Passo 2: Entender a causa raiz do problema – o problema por detrás do problema *28*

Atacando a Causa Raiz 29

Passo 3: Identificar Stakeholders e Usuários *30*

Passo 4: Definir a Fronteira da Solução Sistêmica *32*

Passo 5: Identificar as restrições que serão impostas à solução *34*

Sumário *36*

Vislumbrando o Futuro *36*

CAPÍTULO 5 ..... **38**

Modelagem de Negócio 38

Propósito da Modelagem de Negócio *39*

Usando Técnicas de Engenharia de Software para Modelar Negócios *39*

Escolhendo a Técnica Correta 39

A Linguagem de Modelagem Unificada 40

Modelagem de Negócio Usando UML 40

Da Modelagem de Negócio aos Modelos de Sistemas *42*

Quando Usar a Modelagem de Negócio *42*

Sumário *43*

Vislumbrando o Futuro *43*

CAPÍTULO 6 ..... **44**

ENGENHARIA DE SISTEMAS DE SOFTWARE INTENSIVOS 44
O que é Engenharia de Sistemas? *44*
Princípios Pragmáticos da Engenharia de Sistemas 45
A Composição e Decomposição de Sistemas Complexos 46
Alocação de Requisitos na Engenharia de Sistemas *47*
Sobre Requisitos Derivados 48
Uma Revolução Silenciosa 49
Quando as Gerações se Colidem: os Anciões Encontram os Jovens Arrogantes 49
Evitando o problema do sistema de chaminés 51
Quando Subsistemas São Subcontratos 51
Fazendo o Trabalho de Corretamente 51
O Caso de Estudo *54*
Necessidades Preliminares do Usuário 54
Análise do Problema 54
HOLIS: O Sistema, Atores e Stakeholders *56*
Engenharia de Sistemas do HOLIS 57
Os Subsistemas do HOLIS 58

SUMÁRIO DA HABILIDADE DE EQUIPE 1 ..... **61**

HABILIDADE DE EQUIPE 2 ..... **62**

ENTENDENDO AS NECESSIDADES DOS USUÁRIOS 62

CAPÍTULO 7 ..... **64**

OS DESAFIOS DA ELUCIDAÇÃO DE REQUISITOS 64
Obstáculos da Elucidação *64*
A Síndrome do "Sim, Mas" 64
A Síndrome das "Ruínas Desconhecidas" *65*
A Síndrome "Usuário e o Desenvolvedor" *66*
Técnicas de Elucidação de Requisitos *67*

CAPÍTULO 8 ..... **68**

AS CARACTERÍSTICAS (FEATURES) DE UM PRODUTO OU SISTEMA 68
Stakeholders e Necessidades do Usuário *68*
Características (Features) *69*
Gerenciando a Complexidade Escolhendo o Nível de Abstração 71
Atributos das Características do Produto 71

CAPÍTULO 9 ..... **73**

ENTREVISTA 73
O Contexto da Entrevista *73*
As Questões livres de contexto 73
A Importância do Contexto da Solução *74*
O Momento da Verdade: A Entrevista *77*
Compilando os Dados de Necessidade (Needs) *77*
O Resumo do Analista: 10 + 10 + 10 ≠ 30 78
O Estudo de Caso 78
Uma Nota sobre Questionários *79*

CAPÍTULO 10 ..... **80**

WORKSHOPS DE REQUISITOS 80
Acelerando o Processo de Decisão *80*
Preparando o Workshop *81*
Vendendo o Conceito 81
Assegurando a Preparação dos Stakeholders Corretos *81*
Logísticas 81
"Material de Aquecimento" 82
Papel do Facilitador *84*
Preparando a Agenda *84*
Executando o Workshop *85*
Problemas e Truques do Ofício 85
Brainstorming e Redução de Idéias 87
Produção e Continuidade 88

CAPÍTULO 11 ..... **89**

BRAINSTORMING E A REDUÇÃO DE IDÉIAS 89

Brainstorming Presencial *90*

Redução de Idéias *91*

Expurgando 92

Agrupando Idéias 92

Definição de Características 93

Priorização 93

Brainstorming Baseado em Web *95*

O Caso de Estudos: O Workshop de Requisitos do HOLIS 2000 ..... *95*

Participantes 95

O Workshop 97

A sessão 97

Análise de Resultados 98

CAPÍTULO 12 ..... **100**

STORYBOARDING 100

Tipos de Storyboards *101*

O que os Storyboards Fazem *102*

Ferramentas e Técnicas para o Storyboarding *103*

Dicas do Storyboarding *103*

Sumário *104*

CAPÍTULO 13 ..... **107**

APLICANDO USE CASES 107

Construindo o Modelo Use-Case *108*

Aplicando Use Cases para Elucidação de Requisitos *109*

Caso de Estudos: Os Use Cases do HOLIS *110*

Sumário *111*

CAPÍTULO 14 ..... **112**

ROLE PLAYING 112

Como Interpretar o Papel *113*

Técnicas Similares ao Role Playing *114*

Roteiro de Acompanhamento 114

Cartões CRC (Classe-Responsabilidade-Colaboração) 114

Sumário *115*

CAPÍTULO 15 ..... **116**

PROTOTIPAÇÃO 116

Tipos de Protótipos *116*

Protótipos de Requisitos *117*

O que Prototipar *118*

Construindo o Protótipo *119*

Avaliando os Resultados *119*

Sumário *120*

SUMÁRIO DA HABILIDADE DE EQUIPE 2 ..... **121**

HABILIDADE DE EQUIPE 3 ..... **122**

DEFININDO O SISTEMA 122

CAPÍTULO 16 ..... **125**

ORGANIZANDO INFORMAÇÕES DE REQUISITOS 125

Organizando Requisitos de Sistemas Complexos de Hardware e Software *126*

Requisitos de Organização para Família de Produtos *129*

Sobre os Requisitos "Futuros" *130*

Requisitos de Negócio e de Marketing versus Requisitos de Produto *130*

O Caso de Estudos *131*

Sumário *132*

CAPÍTULO 17 ..... **133**

O DOCUMENTO DA VISÃO 133

Componentes do Documento da Visão *133*

O Documento da "Visão Delta" *137*

Documento da Visão para o Release 1.0 137

Documento da Visão para a Versão 2.0 138

O Documento da Visão Delta num Ambiente de Sistema Legado 140

CAPÍTULO 18 ..... **141**

O CAMPEÃO 141

O Papel do Campeão do Produto *141*

O Campeão do Produto num Ambiente de Produto de Software *142*

O Campeão do Produto numa Empresa de IS/IT *145*

SUMÁRIO DA HABILIDADE DE EQUIPE 3 ..... **147**

HABILIDADE DE EQUIPE 4 ..... **148**

GERENCIANDO O ESCOPO 148

CAPÍTULO 19 ..... **150**

O PROBLEMA DO ESCOPO DE PROJETO 150

A Difícil Questão *153*

CAPÍTULO 20 ..... **154**

ESTABELECENDO O ESCOPO DE PROJETO 154

O Baseline de Requisitos *154*

Definindo as Prioridades *155*

Avaliando o Esforço *156*

Adicionando o Elemento Risco *158*

Reduzindo o Escopo *159*

Uma Estimativa Inicial Razoável 159

O Caso de Estudos *161*

CAPÍTULO 21 ..... **164**

GERENCIANDO O SEU CLIENTE 164

Engajando Clientes para Gerenciar Seu Escopo de Projeto *164*

Comunicando os Resultados *164*

Negociando com o Cliente *165*

Gerenciando o Baseline *166*

Mudança Oficial 167

Mudança Não-oficial 167

CAPÍTULO 22 ..... **168**

GERENCIAMENTO DE ESCOPO E MODELOS DE PROCESSO DE DESENVOLVIMENTO DE SOFTWARE 168

O Modelo Cascata *168*

O Modelo Espiral *170*

A Abordagem Iterativa *172*

Fases do Ciclo-de-Vida 173

Iterações 173

Workflows 174

O que fazer, O que fazer *175*

SUMÁRIO DA HABILIDADE DE EQUIPE 4 ..... **176**

HABILIDADE DE EQUIPE 5 ..... **177**

REFINANDO A DEFINIÇÃO DO SISTEMA 177

CAPÍTULO 23 ..... **179**

REQUISITOS DE SOFTWARE 179

Capítulo 1

O Problema da Pedra (por Ed Yourdon)

Ponto chave

- *Stakeholder* é alguém que tem interesse no sistema de software que será desenvolvido, ou é alguém que é afetado pelo sistema durante ou após o seu desenvolvimento.

Um de meus estudantes resumiu o assunto discutido neste volume como o problema da "pedra". Ela trabalha como engenheira de software num laboratório de pesquisa, onde seus clientes de projeto normalmente dão a missão a qual ela descreve como "Traga-me uma pedra". Mas quando você lhe entrega a pedra, o cliente diz: "Sim, mas na verdade, o que eu realmente queria era uma *pequena* pedra *azul*". Ao entregar uma pequena pedra azul, verifica se que o que o cliente realmente desejava era uma pequena pedra *esférica* e azul.

No final, concluiu-se que, o que o cliente estava querendo era uma pequena pedra de mármore azul – talvez ele não estivesse seguro do que estava querendo, mas um pequeno mármore azul! Bem, talvez quem sabe, um pequeno olho de gato azul, de mármore também teria servido. Provavelmente ele tenha mudado o seu desejo sobre o que queria, entre a entrega da primeira pedra (grande) e a terceira (pequena e azul).

A cada encontro subseqüente com o cliente, é comum que o desenvolvedor pergunte: "O que você quer fazer com isto?". O desenvolvedor fica frustrado porque ele pensou em algo totalmente diferente quando realizou o árduo trabalho de produzir uma pedra com as características prescritas; o cliente fica igualmente frustrado porque, mesmo que ele tenha encontrado dificuldades para articular o que ele queria, ele está convencido de que expressou seus desejos claramente. O desenvolvedor apenas não entendeu!

Para complicar mais ainda, em muitos projetos reais, mais que dois indivíduos estão envolvidos. Além do cliente e o do desenvolvedor – que podem, naturalmente, ter diferentes nomes e títulos – existem provavelmente o pessoal de marketing, o pessoal de testes e garantia de qualidade, gerentes de produtos, gerente geral, e uma variedade de "stakeholders" (envolvidos) que no seu dia-a-dia serão afetados pelo desenvolvimento do novo sistema.

Todo esse pessoal fica frustrado com o problema de especificar uma "pedra" aceitável, principalmente porque, normalmente, não há tempo suficiente no mundo atual tão competitivo, onde as rápidas mudanças no mundo dos negócios não permitem gastar, por exemplo, 2 anos no "projeto da pedra" e, no final, ter que refazê-lo tudo novamente.

Embora existam dificuldades suficientes ao lidamos com artefatos físicos e tangíveis como a pedra, isso pode se complicar mais ainda. Atualmente, as organizações de negócio e agências governamentais são do tipo "informações-intensivas", de tal forma que, mesmo que elas nominalmente estejam no negócio de construir e vender pedras, existe uma boa chance de que a pedra contenha um sistema computacional embutido. Mesmo que isso não ocorra, existe uma boa chance de que o negócio precise de sistemas elaborados para manter as informações sobre as vendas de pedras, clientes que compram

pedra, fornecedores, concorrência, e todas as outras informações que são necessárias para manter competitivo o negócio de vendas de pedras via e-commerce.

Os sistemas de software, devido a sua natureza, são intangíveis, abstratos, complexos e – teoricamente ao menos – estão infinitamente sujeitos a mudanças. Assim, se o cliente começar a articular requisitos vagos para um "sistema de pedras", ele o faz supondo, normalmente, que ele poderá esclarecer, mudar, e fornecer detalhes a posteriori. Seria maravilhoso se desenvolvedores – e quaisquer outras pessoas envolvidas na criação, teste, implantação, e manutenção do sistema de pedras – pudessem realizar esta tarefa em tempo zero e em custo zero; mas isso não acontece assim.

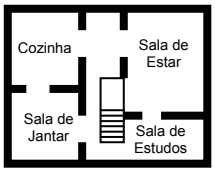
De fato, isso não acontece nunca: Mais da metade dos projetos de sistemas de software que estão, atualmente, em andamento, já ultrapassaram substancialmente o custo e o cronograma previstos, e 25% a 33% desses projetos serão cancelados antes que estejam finalizados, normalmente após consumirem grandes somas de dinheiro.

Prevenir tais falhas e fornecer uma abordagem racional para construir sistemas que o cliente deseja é o objetivo deste volume. É importante advertir que este volume não trata de assuntos de programação e, muito menos escrito somente para desenvolvedores de software. Este volume trata do assunto Gerenciamento de Requisitos para aplicações de software complexas. *Assim, este volume foi escrito para todos os membros da equipe de desenvolvimento – analistas, desenvolvedores, testers, pessoal da Garantia de Qualidade (QA), gerentes de projetos, pessoal de documentação, entre outros, assim como para aqueles membros da equipe de clientes – usuários e outros stakeholders, da área de marketing e gerenciamento* – enfim, todos que, de fato, tenham necessidade e desejos de contribuir com a solução de requisitos.

Irá se descobrir quão crucial é que membros de ambas as equipes, incluindo pessoas da equipe externa que não são da área técnica, dominem as habilidades necessárias para definir e gerenciar com sucesso o processo de requisitos para o novo sistema – isso por uma única razão: são eles que criam inicialmente os requisitos e que, no final, determinam o sucesso ou a falha do sistema. Os programadores heróis e solitários são figuras do passado: Eles podem descansar em paz.

Uma simples comparação: Um empreiteiro não precisa ser convencido de que são necessárias várias conversas, cuidadosamente orquestradas com o dono da casa, para que não seja construída uma casa com dois quartos, quando o dono queria três. Mas é igualmente importante que esses requisitos sejam discutidos e negociados com as autoridades governamentais, responsáveis pelo código de construção civil e leis de zoneamento, e conversar com os moradores das casas vizinhas antes de podar algumas árvores sob a propriedade onde a casa será construída.

Os agentes fiscais da construção civil e das leis de zoneamento, bem como os vizinhos estão entre os stakeholders que, junto com a pessoa que pretende pagar pela casa e morar, irão determinar se a casa, após sua construção, atenderá ao conjunto total de requisitos. É claro, também, que muitos stakeholders, tais como vizinhos e agentes fiscais, não serão os moradores dessa casa (ou usuários do sistema), e parece igualmente óbvio que as perspectivas desses stakeholders sobre o que é uma casa (ou sistema) de qualidade pode variar bastante.



Vale ressaltar que, estamos discutindo aplicações de software neste volume, não sobre casas e pedras. Os requisitos de uma casa podem ser descritos, ao menos em parte, por um conjunto de desenhos e projetos de engenharia; da mesma forma, um sistema de software pode ser descrito com diagramas e modelos. Mas apenas os desenhos da casa são usados como mecanismo de comunicação e negociação entre pessoas leigas (advogados, agentes fiscais e vizinhos abelhudos) e engenheiros.

Assim, diagramas técnicos associados ao sistema de software também podem ser criados com o objetivo de permitir que qualquer pessoa possa entendê-los.

Muitos requisitos cruciais e importantes não necessitam de quaisquer diagramas; por exemplo, potenciais compradores da casa podem escrever requisitos em português comum: "Minha casa deve ter três quartos e deve ter uma garagem grande o suficiente para acomodar dois carros e seis bicicletas". Poderá ser verificado, ao longo deste volume, que requisitos de um sistema de software, em sua grande maioria, podem ser escritos em português comum.

Muitas das habilidades que a equipe precisará para se tornar um especialista, a fim de vencer desafios, pode também ser descrito em termos práticos, como conselhos de senso comum. Por exemplo, para um novato em construção de casas, o seguinte conselho poderia ser dado: "Assegure-se de conversar com os agentes fiscais antes de cavar a fundação da casa e não após preenchê-lo com cimento, construir paredes e levantar o teto". Num projeto de desenvolvimento de software, conselhos similares podem ser fornecidos: "Assegure-se de fazer perguntas corretas; assegure-se de priorizar os requisitos e não permita que clientes lhe digam que 100% dos requisitos são críticos, porque assim, não dará tempo para finalizá-los antes de prazo previsto".

Capítulo 2

Introdução ao Gerenciamento de Requisitos

Pontos chaves

- Um requisito é uma capacidade que o sistema deve apresentar.
- Gerenciamento de requisitos é um processo sistemático de elucidar, organizar e documentar requisitos de sistemas complexos.
- Nosso problema está em entender os problemas dos usuários, sua cultura, sua linguagem, e construir sistemas que atendam as suas necessidades (*need*).
- Uma característica (*feature*) é um serviço que o sistema fornece para atender um ou mais necessidades dos stakeholders.
- Um use case descreve uma seqüência de ações que, quando executada pelo sistema, produz resultados importantes para o usuário.

Uma das 6 Melhores Práticas da Engenharia de Software, Gerenciamento de Requisitos, justifica o foco no gerenciamento de requisitos. Mas antes de explanar as várias técnicas e estratégias para gerenciar requisitos, é necessário fornecer algumas definições e exemplos. É necessário definir o que se entende por requisitos.

Definições

O que é um Requisito?

Embora várias definições para requisitos de software tenham sido usadas durante anos, a definição dada por Dorfman e Thayer (1990) é perfeitamente cabível:

- Uma capacidade que o software que o usuário precisa a fim de resolver um problema e atingir seu objetivo.
- Uma capacidade de software que deve ser atendida ou possuída por um sistema ou componentes do sistema para satisfazer um contrato, padrão, especificação, ou outros documentos formalmente impostos.

Esta definição pode parecer um tanto vago, mas por ora esta definição será o suficiente.

O que é o Gerenciamento de Requisitos?

Requisitos definem as capacidades que o sistema deve apresentar. Normalmente, a adequação ou não do sistema ao conjunto de requisitos determina o sucesso ou o fracasso dos projetos. Assim, é importante descobrir quais são os requisitos do sistema, descrevê-los, organizá-los, e rastrear os eventos que provocam as suas mudanças. Dessa forma, define-se Gerenciamento de Requisitos como:

Uma abordagem sistemática de elucidar, organizar e documentar os requisitos do sistema; e

Um processo que estabelece e mantém um contrato entre cliente e a equipe de projeto sobre as mudanças de requisitos do sistema.

- Qualquer um que já tenha se envolvido com o desenvolvimento de sistemas de software complexo – seja na perspectiva de cliente ou de desenvolvedor – sabe que a habilidade mais importante é a habilidade de *elucidar* requisitos de usuários e stakeholders.
- Uma vez que centenas, se não milhares, de requisitos podem estar associados a um sistema, é importante *organizá-los*.
- Já que o ser humano não possui a capacidade de manipular mais do que 12 peças de informações simultaneamente, é importante *documentar* os requisitos para garantir a comunicação efetiva entre os vários stakeholders. Os requisitos devem ser registrados em algum meio acessível: documento, modelo, base de dados, ou em uma lista sobre o quadro negro.

Mas o que isso têm a ver com o gerenciamento de requisitos? O tamanho e a complexidade de um projeto são os principais fatores aqui: ninguém se preocuparia em gerenciar requisitos num projeto com duas pessoas e que tivesse apenas 10 requisitos para serem atendidos. Mas ao tentar atender a 1.000 requisitos – num pequeno produto de software a ser adquirido – ou a 300.000 requisitos – num Boeing 777 – torna-se óbvio que surgirão problemas para organizar, priorizar, controlar o acesso, e fornecer recursos para esses vários requisitos.

- Quais membros da equipe de projeto são responsáveis pelo requisito (#278), e quem tem a permissão de modificá-lo ou removê-lo?
- Se o requisito #278 for modificado, quais outros requisitos serão afetados?
- Como assegurar que os códigos escritos e respectivos casos de testes, desenvolvidos para satisfazer o requisito #278, serão plenamente atendidos?

As atividades associadas e que respondem a estas e outras questões são as que constituem o Gerenciamento de Requisitos.

O Gerenciamento de Requisitos não é nada novo, não é algo que tenha sido inventada por mero capricho; ele é formado por atividades do "senso comum" as quais muitas organizações de desenvolvimento afirmam realizar de uma forma ou de outra. Mas, normalmente, é realizada de uma maneira informal e persistindo os problemas de um projeto a outro, e algumas das atividades chaves são provavelmente negligenciadas ou levemente alteradas devido às pressões e políticas associadas à maioria dos projetos de desenvolvimento. Portanto, o gerenciamento de requisitos pode ser entendido como um conjunto de técnicas e processos organizados, padronizados e sistematizados com o objetivo de lidar com requisitos de um projeto significativamente complexo.

Existem vários esforços no sentido de organizar e formalizar processos, tais como o SEI-CMM (*Software Engineering Institute's Capability Maturity Model*) e os padrões de gerenciamento da qualidade ISO 9000. As visões de Gerenciamento de Requisitos da SEI-CMM e ISO 2000 serão discutidas no Apêndice D.

Aplicação das Técnicas de Gerenciamento de Requisitos

Tipos de Aplicações de Software

No início, sugeriu-se que as aplicações de software podem ser categorizadas como:

- Sistemas de informação e outras aplicações desenvolvidas para serem utilizadas dentro de uma empresa, tais como Sistemas de Folha de Pagamento utilizados para calcular o salário líquido para o próximo mês. Esta categoria é a base para a indústria de sistemas de informação / tecnologia de informação, ou IS/IT (*Information system / information technology*).
- Software desenvolvido e vendido como produto comercial, tal como o Processador de Textos utilizado para escrever este capítulo. Empresas que normalmente desenvolvem este tipo de software são chamadas de fornecedores de software independentes, ou ISV (*Independent software vendors*).
- Software que são executados em computadores embutidos em outros periféricos, máquinas, ou sistemas complexos, tais como aqueles que estão contidos nos aviões; telefones celulares; e em alguns automóveis de luxo. Este tipo de software é chamado de aplicações de sistemas embutidos, ou simplesmente de aplicações embutidas.

A natureza das aplicações desenvolvidas nestes três tipos de sistemas é extremamente adversa. Elas podem consistir de 5.000.000 linhas de programas COBOL, executados num mainframe e desenvolvidos ao longo de muitos anos por 50 a 100 indivíduos. Elas podem consistir de 10.000 linhas de código em Java, executados sobre uma aplicação cliente Web e escrita em apenas um ano por uma equipe de uma a duas pessoas. Ou elas podem consistir de 1.000.000 de linhas de código C de tempo extremamente crítico e executada sobre um sistema de telefonia complexo, de tempo-real.

Pode se afirmar que as técnicas de Gerenciamento de Requisitos apresentadas neste volume podem ser aplicadas em qualquer um desses três tipos de sistemas. Muitas dessas técnicas são independentes do tipo de aplicação; outros podem necessitar de um refinamento para que possam ser aplicadas no contexto específico da aplicação. Para elevar o entendimento pelo leitor, serão fornecidos exemplos para ilustrar a aplicação dessas diversas técnicas.

Aplicações de Sistemas

O Gerenciamento de Requisitos pode também ser aplicado no desenvolvimento de quaisquer outros tipos de sistemas. Várias técnicas apresentadas neste volume podem ser úteis no gerenciamento de requisitos de sistemas arbitrariamente complexos, contendo subsistemas mecânicos, subsistemas computacionais, subsistemas químicos e peças inter-relacionadas. Claramente, esta é uma disciplina muito ampla e ponderações devem ser feitas para que traga resultados úteis aos membros das equipes de software. Assim, o foco estará num processo e nas técnicas especificas de gerenciamento de requisitos que possam ser aplicadas diretamente nos três tipos de aplicações de software descritos anteriormente: *IS/IT, ISV* e *sistemas embutidos*.

O Mapa da Mina

Já que foi dada a largada para a jornada de se *desenvolver software com qualidade – dentro do prazo e cronograma previstos – e que atenda as reais necessidades dos clientes*, seria muito útil apresentar um mapa descrevendo este território. Não será fácil, uma vez que, durante essa jornada em particular, diversas pessoas que falam diferentes linguagens podem ser encontradas pelo caminho. Muitas dúvidas irão aparecer:

- Isso é uma necessidade ou um requisito?
- Isso é uma coisa que **deve ter** ou que **seria bom ter**?
- Isso é uma declaração do problema ou uma declaração de uma solução?
- Isso é um objetivo do sistema ou um requisito contratual?
- Terá que ser programado em Java? Então quem será o programador?
- Quem é que não gostou do novo sistema e onde está a pessoa que estava aqui antes?

A fim de caminhar com segurança através desse território, será necessário conhecer onde estaremos em alguns pontos do tempo, quem serão as pessoas que encontraremos pelo caminho, a língua que eles falam, e que informações devemos obter dessas pessoas para completar com sucesso a nossa jornada. A jornada começa na "ilha do problema".

O Domínio do Problema

Muitas jornadas de requisitos que obtiveram sucesso começaram com uma visita à ilha do problema. O *domínio do problema* é a casa dos verdadeiros usuários e stakeholders, pessoas cujas necessidades devem ser atendidas a fim de poder construir o sistema perfeito. É a casa das pessoas que necessitam da pedra, ou de um sistema para dar entrada aos pedidos de venda, ou um sistema de gerenciamento de configurações bom o suficiente para vencer a concorrência. Provavelmente, essas pessoas não são como nós. As experiências técnicas e econômicas são diferentes dos nossos, eles falam siglas engraçadas, eles vão a festas diferentes e tomam bebidas diferentes, eles não vestem camisetas para trabalhar, e possuem motivações que são estranhos e impenetráveis. (O quê? Você não gosta do filme *Star Trek*?).

Em raras ocasiões, eles são como nós. São programadores procurando por uma nova ferramenta ou desenvolvedores de sistemas que pediram que você desenvolvesse uma parte do sistema. Nesses raros casos, esta parte da jornada talvez seja fácil, mas pode também ser muito mais difícil.

Mas normalmente, esse não é o caso, e nós nos encontramos na ilha do usuário alienígena. Esses usuários têm negócios ou problemas técnicos que necessitam que nós ajudemos a resolver. Assim, o *nosso* problema está em entender o *seu* problema, dentro de *sua* cultura e *sua* linguagem, para que seja possível construir o sistema que atenda a *suas* necessidades. Como esse território pode parecer nublado, o domínio do problema é representado como uma nuvem cinza. Isso foi feito propositadamente para nos lembrar e nos assegurar de que nós visualizamos claramente todos os casos dentro do espaço do problema.

Domínio do Problema

Dentro do domínio do problema, usamos um conjunto de habilidades de equipe como o mapa e o compasso para entendermos o problema que terá que ser resolvido. Enquanto

estivermos aqui, precisaremos adquirir um entendimento do problema e as necessidades que devem ser atendidas para atacar esse problema.

Necessidades dos Stakeholders

É também de nossa responsabilidade entender as necessidades dos usuários e de outros stakeholders cujas vidas serão afetadas pela nossa solução. Quando nós elucidarmos essas necessidades, nós os colocaremos numa pequena pilha chamada Needs (necessidades) dos stakeholders, a qual representamos como uma pirâmide.

Caminhando em Direção ao Domínio da Solução

Felizmente, a jornada através do domínio do problema não é necessariamente difícil, e os artefatos não são muitos. No entanto, mesmo com essa pequena quantidade de dados, esse é o trecho da jornada no qual nós devemos estar mais bem preparados para fornecer uma solução para o problema. No espaço da solução, nós focalizamos na definição de uma solução para o problema do usuário; este é o mundo dos computadores, programação, sistemas operacionais, redes e dispositivos de processamento. Aqui, nós podemos aplicar diretamente, todas as habilidades que nós aprendemos.

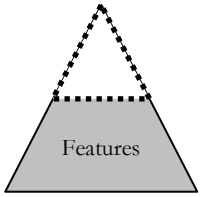
Características do Sistema

Inicialmente, será útil declarar o que aprendemos no domínio do problema e como pretendemos resolvê-lo através da solução. Isso não é muito difícil e deve consistir de itens como:

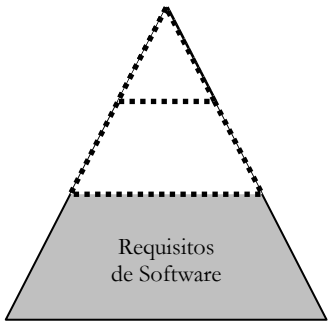
- "O carro terá quadros de potência"
- "Gráficos de análise de defeitos fornecerá um visual significativo para estimar o progresso"
- "Entrada dos pedidos de vendas via Web"
- "Ciclos de repetição automática"

São descrições simples, na linguagem do usuário, as quais serão utilizadas como rótulos para comunicar ao usuário sobre como o nosso sistema irá atacar o problema. Esses rótulos se tornarão parte da linguagem diária, e será gasta muita energia para defini-los, debatê-los e priorizá-los. Nós chamamos esta descrição de "*features*" (características) do sistema que será construído.

> *Uma feature é um serviço que o sistema fornece para atender um ou mais necessidades dos stakeholders.*



Graficamente, representamos as características como a base para pirâmide das necessidades.



Requisitos de Software

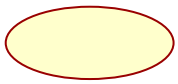
Tendo estabelecido o conjunto de características em comum acordo com o cliente, nós partimos para definir os requisitos mais específicos necessários para a solução. Se construirmos um sistema que atenda a esses requisitos, podemos estar certos de que o sistema que desenvolvemos irá apresentar as características que prometemos. Uma vez

que cada uma dessas características atenda a um ou mais necessidades dos stakeholders, teremos atendido todas as necessidades diretamente na solução.

Esses requisitos mais específicos são os requisitos de software. Representamos esses requisitos de software da mesma forma que fizemos para representar as características.

Uma Introdução aos Use Cases

Um construtor chave irá nos ajudar no final de nossa jornada. Este construtor chave é o *use case*, o qual usamos de várias maneiras através desde volume. De forma simples, um use case descreve uma *seqüência de ações, executadas pelo sistema, e que produz um resultado útil para um usuário*. Em outras palavras, um use case descreve uma série de interações usuário-sistema as quais ajudam o usuário a executar alguma tarefa. Representamos o use case como ícone oval com o nome do use case. Por exemplo, se quisermos descrever um use case de acordo com a intenção do usuário de simplesmente acender ou apagar uma lâmpada, nós podemos chamá-lo, por exemplo, de "Controlar lâmpada", e o colocamos esse nome abaixo do ícone oval.



Controlar lâmpada

Resumo

Agora, vamos olhar o mapa que acabamos de construir. Na figura 2-1, você pode ver que fizemos uma transição sutil, porém importante, em nossa forma de pensar. Nós caminhamos do domínio do problema, representado pela nuvem, passamos pelas necessidades que descobrimos do usuário, até chegarmos na definição de um sistema a qual constitui no domínio da solução, representado pelas características do sistema e pelos requisitos do software, os quais irão dirigir o projeto e a implementação do novo sistema. Mais ainda, fizemos de tal forma que conseguimos assegurar que entendemos o problema e as necessidades do usuário antes de antever ou definir a solução. Este mapa da mina, junto com suas importantes diferenças, irão continuar a serem importantes durante o restante deste volume.

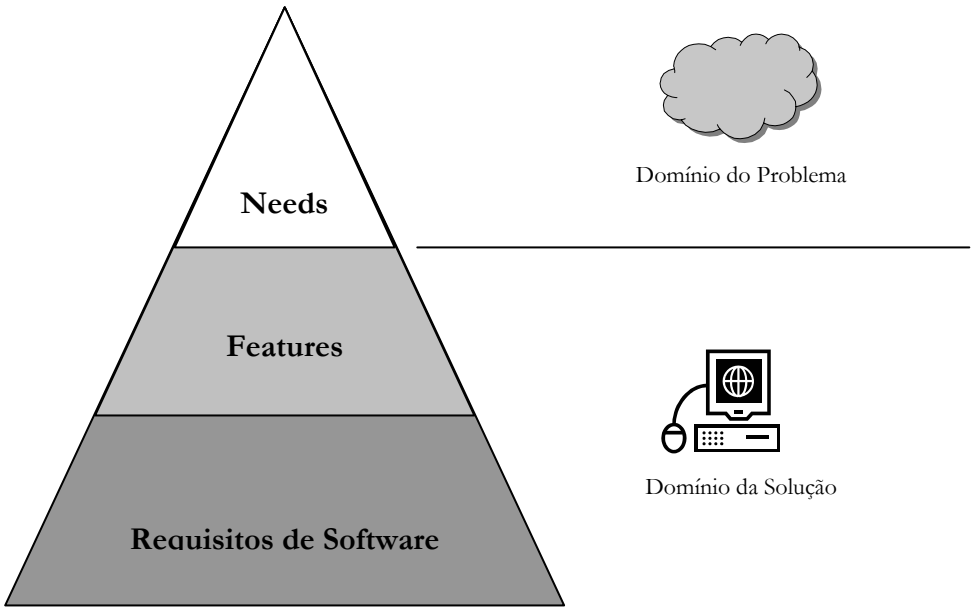


Figura 2-1 Visão global dos domínios do problema/solução

Capítulo 3

A Equipe de Software

"A programação de computadores é uma atividade de negócio" (Weinberg 1971)

Pontos chaves

- O gerenciamento de requisitos afeta todos os membros da equipe, embora de maneiras diferentes.
- O gerenciamento efetivo de requisitos somente poderá ser realizado por uma equipe de software efetiva.
- São necessárias seis habilidades de equipes para o gerenciamento de requisitos.

As pessoas optam pela profissão de desenvolver software por razões variadas. Alguns lêem a revista *Popular Science* e *Popular Mechanics* em casa, realizam cursos de programação de computadores no colégio, estudam Engenharia ou Ciência da Computação na faculdade e por isso, direcionam suas vidas para seguir especificamente o caminho da tecnologia. Para outros, devido à capacidade de demonstrar e de realizar descobertas; encontraram um lugar no tempo e no espaço quando a necessidade por software era premente; e acabaram por se comprometer, gradualmente, com esta área em tempo integral.

Em muitos casos, a atração por tecnologia manteve a chama acesa. Nós amamos bits e bytes, os sistemas operacionais, os bancos de dados, as ferramentas de desenvolvimento, atalhos de teclado e as linguagens de programação. Quem mais, senão os desenvolvedores de software, poderiam ter criado o sistema operacional UNIX? Nosso foco está na tecnologia; essa é a nossa motivação. Talvez pela tendência genética inata ou talvez por não ter assistido à todas as aulas "bobas" na faculdade – psicologia, sociologia, ou pior, Português! – nós geralmente focamos menos nas pessoas de nosso negócio e muito mais em bits e bytes. Nós tendemos a não participar de festas, e alguns de nós temos problemas em se relacionar com pessoas fora do trabalho, onde não existe sustentação tecnológica comum que possam servir de base para uma discussão.

Como conseqüência desse comportamento, surgiram ferramentas de natureza monousuária utilizada para desenvolver aplicações de tamanho limitado, fazendo com que o desenvolvimento de software se tornasse, cada vez mais, numa atividade individual. Os programadores definiam, projetavam, escreviam e, normalmente, testavam seus próprios trabalhos. Às vezes, *testers* eram alocados para ajudá-los nessa terrível tarefa, mas o foco era, claramente, a atividade individual. Programadores heróis era um paradigma comum.

Desenvolvimento de Software como uma Atividade de Equipe

"*O Desenvolvimento de Software transformou-se num esporte de equipe*". (Booch, 1998)

Em algum ponto, houve a virada. Porquê? Watts Humphrey (1989) observou que:

"A história do desenvolvimento de software revela o aumento em escala".

"*a história do desenvolvimento de software revela o aumento em escala. Inicialmente, alguns indivíduos podiam manipular pequenos programas; o trabalho em pouco tempo cresceu além de suas capacidades. Então, equipes de uma ou duas dúzias de pessoas foram usadas, mas o sucesso era imprevisível. Ao mesmo tempo em que as organizações resolviam problemas para pequenos sistemas, a escala de nossos trabalhos continuaram a crescer. Atualmente, grandes projetos normalmente necessitam de trabalho coordenado de várias equipes.*"

O processo de gerenciamento de requisitos afeta todos os membros da equipe, embora de maneiras diferentes.

Hamphrey observou que a complexidade ultrapassa a nossa habilidade de resolver problemas intuitivamente. Por exemplo, estamos envolvidos num projeto de requisitos que afeta simultaneamente, aproximadamente 30 produtos de uma grande família de produtos. Os requisitos que são gerados influenciam, em tempo real, software que estão sendo escritos por mais de 400 programadores distribuídos em diversas localizações. O sucesso deste projeto depende da coordenação intensa de uma "equipe de equipes", todas trabalhando com uma metodologia comum para atender os desafios impostos pelos requisitos.

O gerenciamento efetivo de requisitos somente pode ser realizado por uma equipe de software efetiva.

O que fazer? Claramente, teremos que trabalhar em equipe e trabalhar bem. Como Boehm (1981) concluiu em seu modelo de estimativas de custo, COCOMO, a capacidade da equipe tem grande impacto na produção de software. Davis (1995b) sustenta em sua discussão sobre produtividade da equipe: "otimizar a produtividade de todos os indivíduos não resulta, necessariamente, na otimização da produtividade da equipe." (página 170). Assim, parece lógico investir algum recurso para tornar a equipe de software mais produtiva.

Habilidades da Equipe de Requisitos para o Gerenciamento Efetivo de Requisitos

Este módulo foi organizado em função de 6 habilidades de equipe, necessárias para uma equipe moderna de software enfrentar os desafios de requisitos.

- Na Habilidade de Equipe 1, **Analisando o Problema**, nós desenvolvemos um conjunto de técnicas que a equipe pode usar para obter entendimento apropriado do *problema* que o novo sistema de software pretende resolver.
- Na Habilidade de Equipe 2, **Entendendo as Necessidades dos Usuários**, nós introduzimos várias técnicas que a equipe pode usar para *elucidar requisitos* a partir dos usuários do sistema e stakeholders. Nenhum conjunto de técnicas irá funcionar em todas as situações; nem será necessário que a equipe se especialize em todas as técnicas. Mas com um pouco de prática e alguma coerência nas seleções e escolhas, a equipe irá elevar sua habilidade de entender as reais necessidades que o sistema deverá atender.
- Na Habilidade de Equipe 3, **Definindo o Sistema**, nós descrevemos o processo inicial pelo qual a equipe converte o entendimento do problema e necessidades

dos usuários para uma definição inicial do sistema que deverá atender tais necessidades.

- Na Habilidade de Equipe 4, **Gerenciamento do Escopo**, nós municiamos a equipe com a habilidade de gerenciar melhor o escopo de um projeto. A final de contas, não importa quão bem entendamos as necessidades, a equipe não pode fazer o impossível, e normalmente será necessário negociar o que será entregue antes que o sucesso possa ser obtido.
- Na Habilidade de Equipe 5, **Refinando a Definição do Sistema**, nós ajudamos a equipe a *organizar* as informações dos requisitos. Além disso, nós introduzimos um conjunto de técnicas que a equipe pode usar para *elaborar* a definição do sistema, ou refiná-la até o nível apropriado para dirigir o projeto e implementação, tal que toda a equipe conheça exatamente qual tipo de sistema será construído.
- Finalmente, na Habilidade de Equipe 6, **Construindo o Sistema Correto**, cobrimos alguns aspectos mais técnicos sobre garantia, verificação, validação, teste e gerenciamento de mudanças de projeto, mostramos como a rastreabilidade pode ser usada para ajudar a assegurar a qualidade resultante.

Membros da Equipe possuem Habilidades Distintas

Uma das coisas mais interessantes sobre equipes é que seus indivíduos têm diferentes habilidades. Afinal de contas, isso é que faz de uma equipe uma equipe. Walker Royce (1998) diz o seguinte:

> *Equilíbrio e cobertura são dois dos aspectos mais importantes para uma equipe de excelência... Uma equipe de futebol precisa ter diversas habilidades; assim como uma equipe de desenvolvimento de software... Raramente uma equipe jogará um bom futebol se não tiver uma boa cobertura, ataque, defesa e um bom técnico. Grandes equipes necessitam de cobertura nas várias posições chaves, com jogadores adequados para cada posição. Mas uma equipe cheia de superstars, cada um se esforçando para marcar gols e competindo para ser o líder da equipe, pode ser preocupante para o equilíbrio da equipe. Jogadores adequados em cada posição e somente alguns poucos líderes preocupados com a equipe normalmente vencem o jogo.*

Na equipe de software, nós esperamos que alguns jogadores tenham provado suas habilidades em trabalhar efetivamente com *clientes*, que outros tenham habilidades de *programação* de software, e que outros tenham habilidades para *testes*. Ainda, outros jogadores da equipe irão precisar ter a habilidade para *projeto e arquitetura*. Muitas outras habilidades são necessárias. Nós esperamos também que a habilidade da equipe de requisitos, em gerenciar requisitos, afete vários membros da equipe de diversas maneiras. Assim, de certa forma, nós esperamos desenvolver todas as habilidades individuais da equipe para ajudar no gerenciamento efetivo de requisitos. Além disso, tentaremos indicar onde podemos alocar cada membro da equipe com habilidade particular necessária.

A Organização da Equipe de Software

O desenvolvimento de software é extremamente complexo, e o domínio no qual nós aplicamos nossas habilidades variam enormemente. Assim, não parece razoável que uma maneira específica de organizar uma equipe de software funcione para todos os casos e nem que seja a mais eficiente do que outras abordagens. Apesar de tudo, certos elementos comuns ocorrem na maioria das equipes de sucesso. Assim, achamos que seja mais importante estabelecer uma equipe hipotética. Porém, ao invés de inventarmos uma equipe ideal, o que seria muito mais fácil e muito acadêmico, decidimos modelar nossa

equipe hipotética considerando uma equipe de desenvolvimento de software existente no mundo real.

A equipe que nós iremos modelar baseia-se numa equipe de software do mundo real que provou ser efetivo em duas grandes áreas: (1) efetividade no gerenciamento de requisitos e (2) cumprimento do cronograma e orçamento. (Naturalmente, nós acreditamos que este seja um relacionamento óbvio de causa-efeito!). Além disso, nós admitimos que muitas outras habilidades devem estar presentes numa equipe que verdadeiramente cumpram sempre esses objetivos. Em nosso caso de estudo, a equipe trabalha para a empresa chamada Lumenations S.A., que irá desenvolver um "Sistema de Automação para Iluminação de Residências", para uso em residências de última geração.

O Caso de Estudo

Nós poderemos atingir um outro objetivo neste volume se pudermos desenvolver um caso de estudo que nós trilhamos a partir dos requisitos iniciais até os requisitos finais. Assim, estaremos aptos não só a aplicar as técnicas que estaremos discutindo em nosso exemplo, mas também, em fornecer exemplos dos produtos do trabalho, ou artefatos, que possam ilustrar os pontos chaves e servir de exemplos para os nossos próprios projetos. O apêndice A deste livro fornece um conjunto de exemplos de artefatos do nosso caso de estudo.

Escopo do Caso de Estudo

A Lumenations S.A. tem sido, por 40 anos, um fornecedor comercial mundial de sistemas de iluminação para uso em produções teatrais amadoras e profissionais. Em 1999, seu rendimento anual atingiu aproximadamente 120 milhões de dólares e as vendas estão caindo. A Lumenations é uma empresa pública e o baixo crescimento nas vendas – não, pior ainda, a falta de qualquer possibilidade razoável de elevar o crescimento em vendas – está afetando negativamente no valor da empresa e o humor de seus acionistas. A última reunião anual foi um tanto desconfortável, pois havia poucas novidades relatadas sobre a perspectiva de crescimento da empresa. O valor de cada ação na última primavera havia chegado a 25 dólares devido a uma enorme quantidade de novos pedidos, mas desde então vem vagarosamente caindo, oscilando em torno de 15 dólares.

A indústria de equipamentos para teatros como um todo tem poucos interessados por novos desenvolvimentos. A indústria encontra-se madura e muito bem consolidada. Uma vez que as ações da Lumenations estão na reserva e sua capitalização é bastante modesta, a sua venda não é uma opção da empresa.

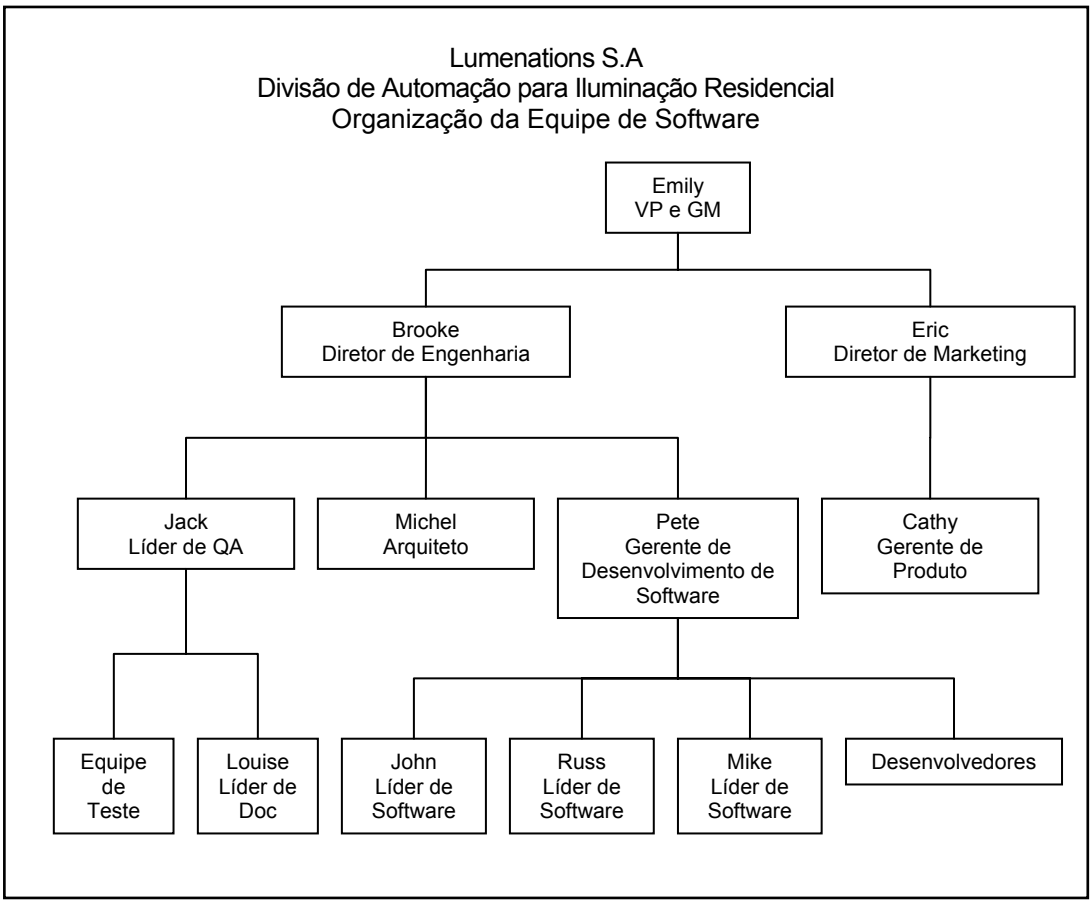
O que é necessário é um novo nicho de mercado, não tão distante do que a empresa faz melhor, mas um que apresente substancial oportunidade de crescimento no rendimento e lucro. Depois de executado um projeto de pesquisa de mercado e gasto muitos dólares para pagar consultores de mercado, a empresa decidiu entrar num novo mercado: Sistema de Automação para Iluminação de Residências de Última Geração. Este mercado aparentemente está crescendo de 25 a 35% ao ano. Melhor ainda, o mercado é imaturo, e nenhuma empresa já estabelecida ocupa a posição de domínio do mercado. O forte canal de distribuição mundial da Lumenations será a real vantagem para ocupar posição no mercado, e os distribuidores estão ávidos por novos produtos. Procurando uma grande oportunidade!

A Equipe de Desenvolvimento do Software HOLIS

O projeto que nós escolhemos desenvolver será o HOLIS, nosso codinome para o novo Sistema de Automação de Iluminação Residencial (HOme Lighting automation System) da Lumenations. O tamanho e escopo da equipe HOLIS é normal. Para o propósito do nosso caso de estudos nós procuramos mantê-lo pequeno, composto por apenas 15 pessoas, mas grande o suficiente para cobrir todas as habilidades necessárias perfeitamente representadas por indivíduos com algum grau de especialização em suas funções. O mais importante é a estrutura da equipe, a qual permite adicionar mais desenvolvedores e testers. A estrutura da equipe HOLIS fornece boa escalabilidade, permitindo elevar o tamanho da equipe para 30 a 50 pessoas, proporcionalmente muito maior do que o sistema HOLIS necessita.

Para atender o novo mercado, a Lumenations configurou uma nova divisão, a Divisão de Automação para Iluminação Residencial. Como a divisão e a tecnologia são novidades para a Lumenations, a equipe HOLIS foi montada num novo local, embora alguns poucos membros da equipe tenham sido transferidos da divisão de iluminação comercial.

A figura abaixo ilustra a organização da equipe de desenvolvimento e as associações entre os membros da equipe. Nós visitaremos cada membro da equipe periodicamente no decorrer deste volume e veremos como eles aplicam suas habilidades para enfrentar os desafios de requisitos do sistema HOLIS.



Sumário

É difícil para alguém racional ir contra a idéia de gerenciar e documentar requisitos de um sistema a fim de assegurar que os resultados irão realmente atender o que o cliente deseja. No entanto as pesquisas demonstram que, como uma indústria, nós freqüentemente realizamos um trabalho pobre. A *falta de retorno dos usuários, requisitos e especificações incompletas, e mudanças de requisitos e especificações* são comumente as causas dos problemas citados nos projetos que falham em atender esses objetivos. E nós sabemos que há um número significativo de projetos de software falham em atender esses objetivos.

Um pensamento comum entre desenvolvedores e clientes é: "mesmo que nós não estejamos seguros dos detalhes que queremos, é melhor iniciar logo a implementação, porque estamos atrasados no cronograma e temos pressa. Nós podemos determinar os requisitos mais tarde". Mas quase sempre esta abordagem, embora bem intencionada, degenera-se para um esforço de desenvolvimento caótico, sem nenhuma segurança sobre o que o usuário realmente deseja ou o que o sistema, assim construído, realmente faz. Com o potencial das ferramentas de prototipação de fácil utilização, existe a percepção de que se os desenvolvedores podem construir um rascunho aproximado do que os usuários desejam num protótipo, o usuário pode indicar as características que necessitam ser adicionadas, removidas ou modificadas. Isso pode funcionar, e é um importante aspecto do desenvolvimento interativo. Mas devido em parte ao extremo custo em corrigir erros de requisitos, este processo precisa fazer parte do contexto global da *estratégia de gerenciamento de requisitos*, caso contrário resultará no caos.

Como saberemos o que o sistema deverá fazer? Como manteremos a trilha do estado atual dos requisitos? Como determinar o impacto de uma mudança? Devido a questões como essas, o gerenciamento de requisitos começou a emergir como uma disciplina de engenharia de software prática. Nós introduzimos uma filosofia confinada ao gerenciamento de requisitos e fornecemos um conjunto de definições que sustentam tais atividades.

Visto que a história do desenvolvimento de software – bem como o futuro, pelo menos até onde podemos prever – revela o aumento em escala, ou seja, a elevação da quantidade de trabalho com o passar do tempo, podemos entender que o problema do desenvolvimento de software deve ser atacado por equipes de software bem estruturadas e treinadas. Na disciplina de gerenciamento de requisitos em particular, todos os membros da equipe estarão eventualmente envolvidas em atividades que auxiliem o gerenciamento de requisitos de projeto. Essas equipes devem desenvolver as habilidades de requisitos para entender as necessidades dos usuários, para gerenciar o escopo da aplicação, e para construir sistemas que atendam as necessidades desses usuários. A equipe de requisitos deve trabalhar *como uma equipe de futebol vencedora*, para enfrentar os desafios que o gerenciamento de requisitos impõem.

A fim de fazer isso, o primeiro passo no processo de gerenciamento de requisitos é assegurar que o desenvolvedor entenda o "problema" que o usuário está tentando resolver. Nós iremos cobrir este tópico nos três próximos capítulos: Habilidade de Equipe 1, Analisando o Problema.

Habilidade de Equipe 1

Analisando o Problema

- **Capítulo 4**: Os Cinco Passos da Análise do Problema
- **Capítulo 5**: Modelagem de Negócio
- **Capítulo 6**: Engenharia de Sistemas – Sistemas de Software-Intensivo

Problema

Em poucos anos veremos um aumento sem precedentes no poder das ferramentas e tecnologias que os desenvolvedores de software usarão para construir aplicações empresariais. Novas linguagens irão elevar o nível de abstração e aumentar a produtividade de atacar e resolver problemas de usuários. A utilização de métodos orientados a objetos tem produzido projetos que são mais robustos e extensíveis. Ferramentas de gerenciamento de versões, gerenciamento de requisitos, análise e projeto, rastreamento de falhas, e testes automatizados, têm ajudado os desenvolvedores de software a gerenciar a complexidade de milhares de requisitos e centenas de milhares de linhas de códigos.

Com a elevação da produtividade dos ambientes de desenvolvimento de software, será mais fácil desenvolver sistemas de software que atendam as reais necessidades de negócio. No entanto, como vimos, as pesquisas demonstram que continuamos sendo desafiados em entender e satisfazer verdadeiramente essas necessidades. Talvez exista uma explicação simples para essa dificuldade: "*o problema por detrás do problema". A equipe de desenvolvimento gasta muito pouco tempo em entender os reais problemas de negócio, as necessidades dos usuários e de outros stakeholders, e a natureza do ambiente na qual suas aplicações devem ter sucesso.* Nós desenvolvedores tendemos a avançar constantemente, fornecendo soluções tecnológicas baseadas sobre um entendimento inadequado do problema a ser resolvido.

Equipes de desenvolvimento tendem a avançar continuamente, fornecendo soluções baseadas em entendimentos inadequados do problema a ser resolvido.

Como esperado, o sistema resultante não atende as necessidades dos usuários e stakeholders. Entre as conseqüências desse insucesso estão: o baixo retorno financeiro de clientes e desenvolvedores do sistema, usuários insatisfeitos, e problemas constantes. Assim, parece óbvio que um investimento incremental na análise do problema irá produzir resultados gratificantes. O objetivo desta habilidade de equipe é fornecer um guia para a análise do problema definindo metas para esta habilidade no desenvolvimento da aplicação.

Nos capítulos seguintes iremos explorar maneiras e meios de definir exatamente o que é um problema. Afinal, se sua equipe não puder definir o problema, será difícil encontrar uma solução apropriada.

Capítulo 4

Os Cinco Passos da Análise do Problema

Pontos chaves

- A análise de problemas é o processo de entender problemas do mundo real, entender as necessidades dos usuários e propor soluções que satisfaçam tais necessidades.
- O objetivo da análise do problema é adquirir maior entendimento do problema a ser resolvido, antes que se inicie o desenvolvimento.
- Para identificar a causa raiz do problema, ou o problema por detrás do problema, pergunte às pessoas diretamente envolvidas.
- Identificar os atores envolvidos no sistema é o principal passo da análise do problema.

Este capítulo descreve as maneiras em que a equipe de desenvolvimento pode entender as necessidades de stakeholders e usuários de um novo sistema ou aplicação do mundo real. Uma vez que, na maioria das vezes, sistemas são construídos para resolver problemas específicos, iremos usar técnicas de análise de problemas para assegurar que entendemos o que é o problema.

Mas devemos também reconhecer que nem todas as aplicações são desenvolvidas para resolver problemas; alguns são construídos para ganhar vantagens sobre oportunidades que o mercado apresenta, mesmo quando a existência de um problema não esteja clara. Por exemplo, aplicações únicas de software, tais como SimCity e Myst, provaram seu valor para aqueles que gostam de jogos de computador e desafios mentais, ou para aqueles que apreciam modelagem e simulação, ou para aqueles que simplesmente querem se divertir jogando em seus computadores. Portanto, ainda que seja difícil dizer qual é o problema que SimCity e Myst resolvem – bem, talvez o problema de "não existirem muitas coisas divertidas para fazer como o seu computador" ou o problema de "ter demasiado tempo livre sem ter o que fazer" – é claro que os produtos fornecem real valor para um grande número de usuários.

Nesse sentido, problemas e oportunidades são ambos, os lados de uma mesma moeda; seu problema é minha oportunidade. Isso é apenas uma questão de perspectiva. Mas como muitos sistemas atendem a algum problema identificável, podemos simplificar a discussão evitando a esquizofrenia do problema/oportunidade, concentrando-nos em apenas um dos lados da moeda: o lado do problema. Afinal de contas, gostamos de pensar que somos os solucionadores de problemas.

Definimos a análise de problemas como:

> *o processo de entender problemas do mundo real, entender as necessidades dos usuários e propor soluções que satisfaçam tais necessidades.*

Dito isso, o domínio do problema deve ser analisado e entendido, e uma variedade de domínios da solução devem ser explorados. Normalmente, várias soluções são possíveis, e nosso trabalho é encontrar a solução que seja o ótimo para o problema a ser resolvido.

Para estarmos aptos a *realizar* a análise de problemas, será útil definir o que é um problema. De acordo com Gause e Weinberg (1989):

> *"um problema pode ser definido como a diferença entre coisas são desejadas e coisas que são percebidas."*

Essa definição pelo menos elimina o problema de desenvolvedores acharem que o real problema é que o usuário não sabe qual é o real problema! De acordo com a definição, se o usuário percebe algo como problema, esse é um real problema digno de ser atacado.

Às vezes, a solução mais simples é uma solução de contorno, ou uma revisão do processo de negócio, ao invés de um novo sistema.

Ainda, com base nesta definição, nosso colega Elemer Magaziner notou que existem várias maneiras de se atacar um problema. Por exemplo, mudar o desejo ou percepção do usuário pode ser a abordagem de melhor custo efetivo. Assim, pode ser uma questão de ajuste e gerenciamento de expectativas, fornecendo soluções de contorno ou aperfeiçoamento incremental para sistemas existentes, fornecendo soluções alternativas que não requeiram o desenvolvimento de novos sistemas, ou fornecendo treinamento adicional. A experiência prática mostra muitos exemplos onde mudar a percepção da diferença tem conduzido a soluções vantajosas, rápidas, baratas e de altíssima qualidade! Como solucionadores de problemas, estamos incumbidos em explorar essas soluções alternativas antes de saltar para a solução de um novo sistema.

Todavia, quando a atividade de encontrar uma solução alternativa para reduzir a diferença entre o percebido e o desejado falhar, estaremos diante de um grande desafio: o de efetivamente reduzir a distância entre o percebido e a realidade. Isso nós devemos realizar definindo e implementando *novos sistemas* que reduzam a diferença entre o *percebido* e o *desejado*.

O objetivo da análise de problemas é adquirir melhor entendimento do problema a ser resolvido antes de iniciar o desenvolvimento.

Como em qualquer exercício de solução de problemas complexos, devemos iniciar tendo um objetivo em mente. O objetivo da análise de problemas é adquirir melhor entendimento do problema a ser resolvido antes de iniciar o desenvolvimento. Os passos que devem ser tomados a fim de alcançar esse objetivo são:

1. Chegar ao acordo sobre a definição do problema.
2. Entender a causa raiz do problema – o problema por detrás do problema.
3. Identificar os stakeholders e usuários.
4. Definir a fronteira da solução sistêmica.
5. Identificar as restrições que serão impostas à solução.

Permita-nos trabalhar cada um desses passos e ver se podemos desenvolver as habilidades de equipe que precisamos para chegar à solução pretendida!

Passo 1: Chegar ao Acordo sobre a Definição do Problema

O primeiro passo é chegar ao acordo sobre a definição do problema as ser resolvido. Uma maneira simples de chegar a esse acordo é *simplesmente descrever o problema e ver se todos concordam*.

Como parte deste processo, normalmente é benéfico entender alguns dos benefícios propostos pela solução, seja cuidadoso assegurando-se de que os benefícios sejam descritos utilizando termos fornecidos pelos clientes/usuários. Descrições realizadas por usuários fornecem fundamento contextual adicional ao real problema. Ao ver os benefícios sob o ponto de vista do cliente, também obtemos a um melhor entendimento do problema do ponto de vista dos stakeholders.

A Declaração do Problema

Você poderá achar útil descrever o seu problema num formato padrão (Tabela 4–1). O preenchimento da tabela, ainda que simples, é uma técnica poderosa para assegurar que todos os stakeholders do seu projeto estejam trabalhando em direção aos mesmos objetivos.

Tabela 4–1 Formato da Declaração do problema

Elementos	Descrição
O problema	Descrever o problema.
afeta	Identificar os stakeholders afetados pelo problema
devido	Descrever o impacto deste problema nos stakeholders e atividades de negócio.
Os benefícios desse	Indicar a solução proposta e listar os principais benefícios.

Gastar tempo para chegar ao acordo sobre o problema a ser resolvido pode parecer um passo pequeno e insignificante, e em muitas circunstâncias, isso é verdade. Mas algumas vezes, não é. Por exemplo, um de nossos clientes, um fabricante de equipamentos, foi realizar uma grande atualização no seu sistema de informação, o qual realiza o faturamento e fornece relatórios financeiros entre a empresa e seus fornecedores. O tema para o novo programa foi "aperfeiçoar comunicações com fornecedores". Dito isso, a equipe tinha iniciado um esforço significativo para o desenvolvimento de um novo sistema.

Um exercício de como chegar ao acordo sobre o problema a ser resolvido foi realizado. A equipe de desenvolvimento definiu uma solução prevendo um novo sistema poderosíssimo que fornecia: relatórios financeiros muito melhores; aperfeiçoamento do faturamento e do formato da fatura, tratamento de pedidos online; e e-mail. Ah, a propósito, a equipe esperava, em algum momento, fornecer a capacidade de transferência de fundos eletronicamente entre a empresa e seus fornecedores.

Durante o exercício de fazer a declaração do problema, a gerência da empresa tinha a oportunidade de fornecer informações. A visão dos gerentes foi substancialmente diferente. O principal objetivo do novo sistema era *transferir fundos eletronicamente para melhorar o fluxo de caixa da empresa.* Depois de uma discussão acalorada, ficou claro que o principal problema a ser atendido pelo novo sistema era a transferência eletrônica de fundos; e-mail e outras características de comunicação com fornecedores foram considerados como algo que "poderia ter". Desnecessário dizer que foi uma substancial reorientação dos objetivos do novo sistema, incluindo uma nova definição do problema que identifica a transferência de fundos como principal problema a ser resolvido. Esta reorientação também disparou o desenvolvimento de uma arquitetura diferente daquele que havia sido previsto, o qual foi completado com a capacidade de segurança consistente com o risco inerente ao banco eletrônico.

Passo 2: Entender a causa raiz do problema – o problema por detrás do problema

Sua equipe pode usar uma variedade de técnicas para obter um entendimento do real problema e suas reais causas. Uma dessas técnicas é a análise "causa raiz", a qual é uma forma sistemática de descobrir a causa raiz, ou a origem, de um problema identificado ou um sintoma de um problema.

Por exemplo, considere um exemplo do mundo real: uma empresa, chamada GoodsAreUs, vende produtos através de catálogos enviados por e-mail; manufatura e vende uma variedade de itens baratos para uso residencial e pessoal. Essa empresa, mobilizada para atacar o problema de baixa lucratividade, utiliza técnicas de gerenciamento de qualidade total (TQM) aprendido em seu programa de qualidade, para solucionar o problema. Baseada em sua experiência, a empresa rapidamente concentrou seu foco para o *custo da não-conformidade*, que é o custo de todas as coisas que estão erradas e que geram perdas, detritos, e outros custos excessivos. Este custo inclui o re-trabalho, detritos, insatisfação de clientes, rotatividade de empregados, e outros fatores que contribuem negativamente. Quando a empresa quantificou seu custo de não-conformidade, suspeitou que as perdas na produção, ou "desperdício", era um dos principais fatores que contribuíam para a baixa lucratividade da empresa.

O próximo passo para descobrir a causa raiz, ou *o problema por detrás do problema de desperdício*, é determinar quais fatores contribuem para o problema de desperdício. O TQM nos ensina que devemos usar o *diagrama espinha de peixe* (veja a Figura 4–1) para identificar o problema por detrás do problema. Em nossa análise específica, a empresa identificou muitas fontes que contribuíam para o desperdício. Cada fonte foi identificada em cada um dos "ossos" do diagrama.

Muito bem, então como você determina a causa raiz? Bem, isso depende. Em muitos casos, isso é apenas uma questão de perguntar às pessoas diretamente envolvidas sobre o que eles acham que é a causa raiz. É incrível como a maioria dessas pessoas conhece o problema por detrás do problema; é apenas isso, mais nada – pelo que nós entendemos de gerenciamento – sempre pergunte antes. Assim, pergunte e pergunte várias vezes.

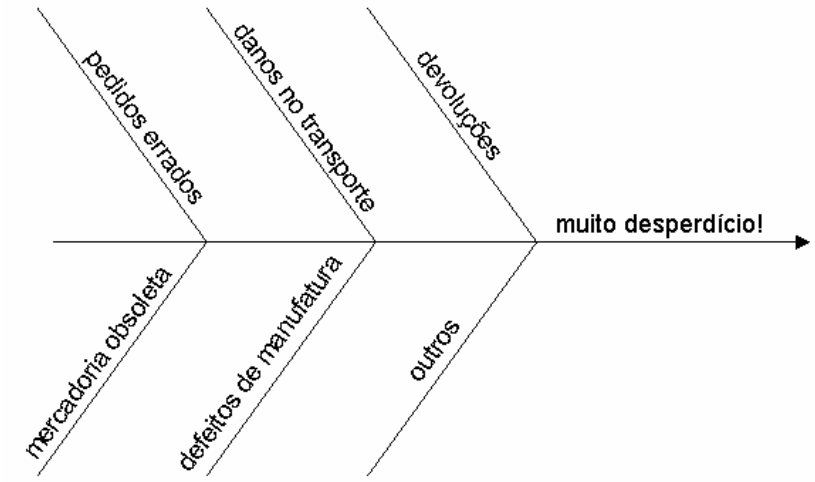


Figura 4–1 Diagrama espinha de peixe de causa raiz

Se o problema é mais sério e levianamente ficarmos perguntando o que pode estar causando o problema, poderá gerar um ambiente desconfortável; pode ser necessário realizar uma investigação detalhada de cada causa do problema e quantificar individualmente seu impacto. Isso pode ser feito utilizando desde um simples brainstorming com participantes que tenham conhecimento do domínio do problema até um pequeno projeto de coleta de dados, ou, potencialmente, até uma investigação científica mais rigorosa. Em muitos casos, o objetivo é quantificar as contribuições prováveis para cada causa raiz.

Atacando a Causa Raiz

Dados de qualidade demonstram que muitas causas raízes não valem a pena serem corrigidas.

Naturalmente, o engenheiro em todos nós gostaria de corrigir todas as causas raízes identificadas nos "ossos" do diagrama. Isso parece coisa certa a fazer. Mas é mesmo? Normalmente, não é; *informações da qualidade mostram, com freqüência, que muitas causas raízes não valem a pena serem corrigidas*, quando o custo de corrigir excede o custo do problema. Então, como vou saber quais causas raízes devo corrigir? *Resposta*: Você deve determinar a importância, ou a contribuição, de cada causa raiz. Os resultados dessa investigação podem ser colocados num gráfico como o de Pareto, ou num simples histograma que exponha visualmente os verdadeiros culpados.

De volta ao nosso exemplo: Suponha que os resultados dos dados obtidos tenham produzido o gráfico da Figura 4–2. Como você pode ver, a equipe descobriu que uma única causa raiz – "pedidos errados" – produziu a metade de todos os desperdícios. Se, por sua vez, o sistema de pedidos existente for um exemplo de um código legado ruim, associado a uma interface de usuário cheia de vícios e não houver tratamento de erros online, esta pode ser a oportunidade de reduzir o desperdício através do desenvolvimento de um novo sistema.

É neste ponto, *e somente neste ponto*, que a equipe irá conseguir justificar o propósito de trocar o sistema de entrada de pedidos existentes. Além disso, a justificativa de custo desse novo sistema pode ser quantificada determinando o custo do desenvolvimento e o retorno deste investimento através da redução do desperdício.

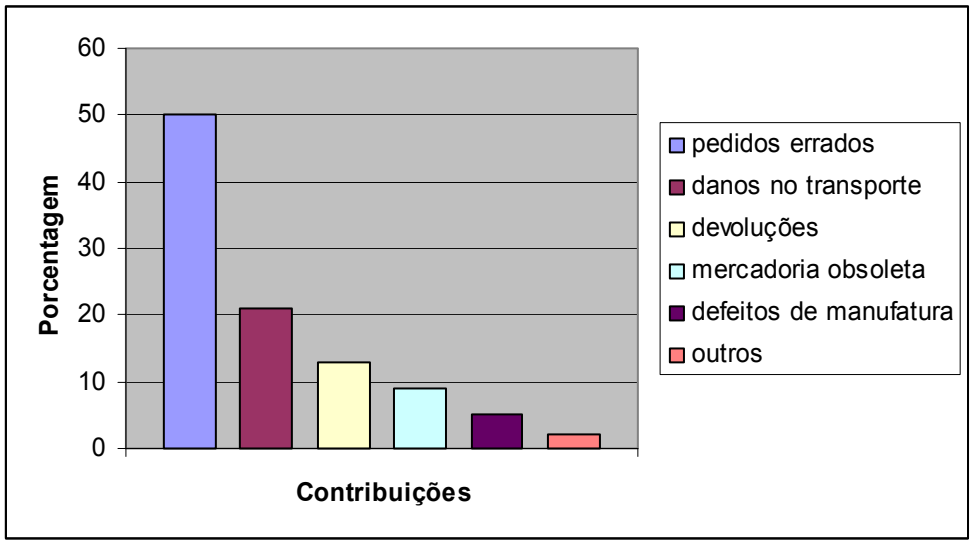


Figura 4–2 Gráfico de Pareto das causas raízes

Posteriormente, a análise do diagrama espinha de peixe pode ser usada para determinar quais tipos específicos de erros contribuem para o problema de pedidos errados de vendas. Esses dados, mais detalhados, podem então ser usados para definir as características do sistema de software para atacar tais erros. Para o nosso propósito, no entanto, podemos finalizar a nossa análise e concordar com a troca do sistema de pedidos que, ao menos é uma solução parcial para o problema de muitos desperdícios.

Uma vez que identificamos "pedidos errados" como uma das causas raízes do problema que vale a pena ser resolvido, nós podemos criar uma declaração do problema para o problema dos pedidos de venda, como ilustrado na Tabela 4–2.

Tabela 4–2 Declaração do problema dos pedidos de venda

Elementos	Descrição
O problema	de pedidos errados
afeta	o pessoal de vendas, clientes, fabricantes, transportadores e serviço de atendimento ao cliente
devido	ao aumento de desperdícios, excessiva manipulação de custos, insatisfação de clientes e baixa lucratividade.
Os benefícios desse	novo sistema que irá atacar o problema são: ▪ Aumento de precisão dos pedidos de venda nos pontos de venda ▪ Aperfeiçoamento de relatórios gerenciais com os dados de venda ▪ E, finalmente, alta lucratividade.

Uma vez descrita, a declaração do problema pode ser divulgada entre os stakeholders para criticarem e tecerem comentários. Quando esse período de receber críticas e comentários estiver finalizado, e a declaração do problema consolidada, a declaração do problema passará a ser uma missão declarada, a qual todos os membros da equipe de projeto terão que ter em suas mentes, para que todos trabalhem para atingir aos mesmos objetivos.

Passo 3: Identificar Stakeholders e Usuários

O entendimento das necessidades dos usuários e stakeholders é o fator chave para o desenvolvimento de uma solução efetiva.

A solução efetiva de qualquer problema complexo quase sempre envolve a satisfação das necessidades de diversos grupos de stakeholders. Os stakeholders, normalmente, possuem várias perspectivas sobre o problema e várias necessidades que esperam que sejam atacadas pela solução. Nós iremos definir um stakeholder como:

> *qualquer um que possa ser substancialmente afetado pela implementação de um novo sistema ou aplicação.*

Muitos stakeholders são usuários do sistema e suas necessidades são fáceis de identificar porque eles estão diretamente envolvidos com a definição e uso do sistema. Porém, alguns stakeholders são apenas usuários indiretos do sistema ou são afetados apenas pelos resultados de negócio que o sistema influencia. Esses stakeholders tendem a serem encontrados além do escopo de negócio, ou nos "arredores" do ambiente de uma particular aplicação. Ainda em outros casos, esses stakeholders serão removidos do ambiente da aplicação. Incluem-se, por exemplo, as pessoas e organizações envolvidas no desenvolvimento do sistema, subcontratados, os clientes dos clientes, agências externas, tais como a FAA (U.S. Federal Aviation Administration) ou a FDA (Food and Drug

Administration), ou outras agências que interagem com o sistema ou com o processo de desenvolvimento. Cada uma dessas classes de stakeholders pode influenciar os requisitos do sistema ou irá de alguma forma estar envolvida com os resultados do sistema.

Stakeholders que não são usuários devem também ser identificados e tratados.

O entendimento de quem são os stakeholders e de suas necessidades particulares é um fator importante para o desenvolvimento de uma solução efetiva. Dependendo do domínio de especialidade da equipe de desenvolvimento, identificar stakeholders pode ser uma tarefa trivial ou não na análise do problema. Freqüentemente, isso envolve simplesmente entrevistar as pessoas que decidem, usuários potenciais, e outras partes interessadas. As seguintes questões podem ser úteis nesse processo:

- Quem são os usuários do sistema?
- Quem é o cliente (aquele que paga) do sistema?
- Quem mais será afetado pelas saídas que o sistema irá produzir?
- Quem irá avaliar e homologar o sistema quando ele for entregue e implantado?
- Existem outros usuários internos ou externos do sistema cujas necessidades devam ser atendidas?
- Quem irá manter o sistema?
- Existe alguém mais?

Em nosso exemplo da substituição do sistema de pedidos, o principal e o mais óbvio dos usuários são os vendedores que entram com os pedidos de vendas. Esses usuários são obviamente stakeholders uma vez que a sua produtividade, conveniência, conforto, desempenho e satisfação no trabalho são afetados pelo sistema. Quais outros stakeholders podem ser identificados?

Outros stakeholders, o supervisor de pedidos de vendas, por exemplo, são diretamente afetados pelo sistema, mas acessam o sistema através de diferentes relatórios e interfaces de usuários. Ainda outras pessoas, o diretor financeiro da empresa, por exemplo, são evidentemente stakeholders, uma vez que o sistema pode ser afetar a produtividade, qualidade e lucratividade da empresa. Para não esquecermos, o diretor de sistemas de informação e membros da equipe de desenvolvimento do sistema são também stakeholders, uma vez que eles são responsáveis pelo desenvolvimento e manutenção do sistema. Eles terão que viver com os resultados, assim como os usuários. A Tabela 4–3 sumariza os resultados da análise de stakeholders e identifica usuários e outros stakeholders do novo sistema de pedidos.

Tabela 4–3 Usuários e Outros Stakeholders do novo sistema

Usuários	Outros Stakeholders
Vendedores	Diretor de SI e equipe de desenvolvimento
Supervisor de Vendas	Diretor Financeiro
Controle da Produção	Gerente de Produção
Pessoal de Faturamento	

Passo 4: Definir a Fronteira da Solução Sistêmica

Uma vez que se tenha chegado ao acordo sobre a declaração do problema e, usuários e stakeholders tenham sido identificados, nós podemos voltar nossa atenção para a *definição do sistema* que poderá ser desenvolvido para atacar o problema. Ao fazer isso, nós entraremos numa importante transição de estados, onde teremos que manter duas coisas em mente: a compreensão do problema e as considerações de uma solução em potencial.

O próximo passo importante é determinar a fronteira da solução sistêmica. A fronteira do sistema define o limite entre a solução e o mundo real que cerca a solução (Figura 4–3). Em outras palavras, a fronteira do sistema descreve um invólucro no qual a solução está contida. As informações, existentes nos formulários de entrada e saída, são repassadas para fora do sistema, para usuários que vivem fora do sistema. Todas as interações com o sistema ocorrem via interfaces entre o sistema e o mundo externo.

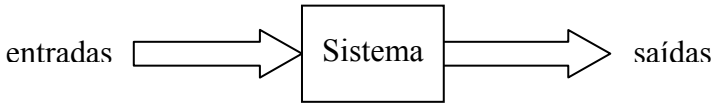


Figura 4–3 A associação entrada/sistema/saída

Nós dividimos o mundo em duas classes:
1. Nosso sistema
2. Coisas que interagem com o nosso sistema

Em outras palavras, se nós tivermos que construir ou modificar algo, esse algo será parte de nossa solução e estará dentro da fronteira; caso contrário, será externo ao nosso sistema. Assim, dividimos o mundo em duas classes interessantes:

1. Nosso sistema
2. Coisas que interagem com o sistema

Ator

Identificar as "coisas que interagem com o nosso sistema" é o mesmo que identificar os "atores de nosso sistema". Afinal de contas, eles possuem um papel para interpretar, que é o de fazer com que o nosso sistema trabalhe. Nós representamos um ator com um simples desenho de um ser humano. Nós definimos um ator como:

alguém ou alguma coisa, fora do sistema, que interage com o sistema.

Uma vez que temos a notação de um ator, podemos ilustrar a fronteira do sistema como ilustra a Figura 4–4.

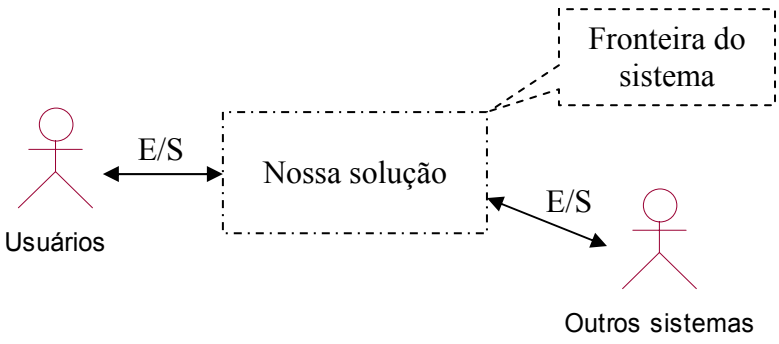


Figura 4–4 Fronteira do sistema

Em muitos casos, a fronteira do sistema é óbvia. Por exemplo, uma copiadora pessoal, conectada a um PC-Windows 2000 tem a fronteira relativamente bem definida. Existe apenas um usuário e uma plataforma. A interface entre o usuário e a aplicação é construída por caixas de diálogo, o qual o usuário utiliza para acessar informações do sistema e configurar qualquer relatório e caminhos de comunicação que o sistema utilize para documentar ou transmitir essas informações.

Em nosso exemplo, sistema de pedidos, que será integrada a um sistema legado, a fronteira não é tão clara. O analista deve determinar se os dados serão compartilhados com outras aplicações, se a nova aplicação estará distribuída entre os vários servidores e clientes, e quem serão os usuários. Por exemplo, o pessoal de produção terá acesso online os pedidos? Existe um controle de qualidade ou funções de auditoria que terá que ser fornecido? O sistema executará num mainframe ou sobre um novo front end cliente/servidor? Relatórios gerenciais específicos terão que ser fornecidos?

Embora possa parecer bastante óbvio, a identificação de atores é o principal passo analítico da análise do problema. Como encontramos esses atores? Aqui estão algumas questões que poderão ajudar a encontrar esses atores:

- Quem irá fornecer, usar, ou remover informações do sistema?
- Quem irá operar o sistema?
- Quem irá realizar as manutenções no sistema?
- Onde o sistema será usado?
- Onde o sistema obtém suas informações?
- Quais outros sistemas externos irão interagir com o sistema?

A partir das respostas para essas questões, o analista poderá criar uma "perspectiva do sistema", um diagrama de blocos que descreve a fronteira do sistema, os usuários, e outras interfaces. A Figura 4–5 fornece uma perspectiva simplificada do sistema para o novo sistema de pedidos.

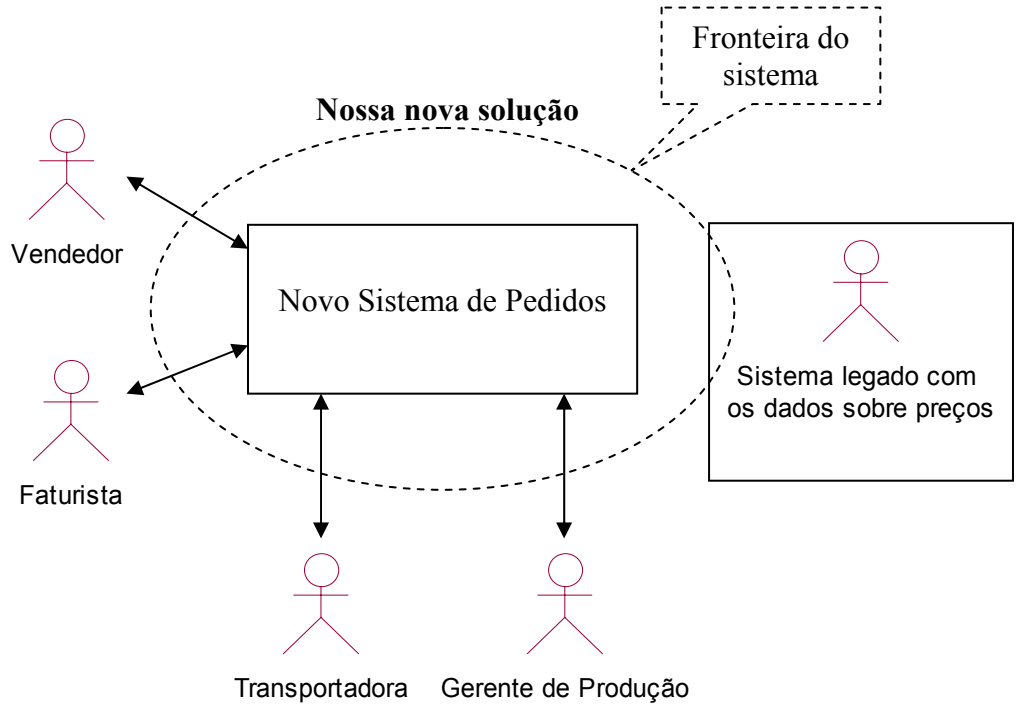


Figura 4–5 Perspectiva do Sistema

A linha tracejada ilustra a fronteira do sistema para a solução proposta. O diagrama ilustra que a maior parte da solução irá ser desenvolvida para um novo sistema de pedidos, mas uma parte do código da solução deverá ser desenvolvida e implantada sob o sistema legado existente.

Passo 5: Identificar as restrições que serão impostas à solução

Restrições são limitações sobre o grau de liberdade que temos em fornecer uma solução.

Antes de gastar trilhões de dólares num esforço bem intencionado que irá revolucionar o estado da arte de sistemas de pedidos, nós devemos parar e considerar as restrições que serão impostas à solução. Definimos uma restrição como:

> *um limite sobre o grau de liberdade que temos em fornecer uma solução.*

Cada restrição tem o potencial para restringir severamente a nossa habilidade de produzir uma solução da forma como estava prevista. Assim, cada restrição deve ser cuidadosamente considerada como parte de um processo de planejamento, pois muitos podem até fazer com que reconsideremos a abordagem tecnológica que previmos inicialmente.

Uma variedade de fontes de recursos deve ser considerada. Isso inclui planejamento retorno de investimento, orçamento de pessoal e equipamentos, assuntos ambientais, sistemas operacionais, banco de dados, sistemas clientes e servidores, assuntos técnicos, assuntos políticos internos à organização, compra de software, políticas e procedimentos da empresa, escolha de ferramentas e linguagens, pessoal e outras fontes de recursos, além de várias outras considerações. Essas restrições podem ser impostas antes mesmo que iniciemos qualquer atividade ("Nenhum novo hardware"), ou teremos que ir atrás e descobri-los.

Para auxiliar nessa descoberta, é útil conhecer o tipo de coisa que devemos procurar. A Tabela 4–4 apresenta fontes potenciais de restrições de sistemas. Responder questões apresentadas nesta tabela ajudará a descobrir as principais restrições que irão afetar nossa solução. Além disso, essa atividade poderá, provavelmente, ajudar na identificação da lógica para a restrição, tanto para assegurar que você entendeu a perspectiva da restrição quanto para você reconhecer quando a restrição não mais se aplica para a nossa solução. Quanto menos restrito, melhor.

Uma vez que as restrições tenham sido identificadas, algumas delas irão se tornar requisitos para o novo sistema ("usar o sistema MRP desenvolvido pelo nosso fornecedor de sistema de conta corrente"). Outras restrições irão afetar recursos, planos de implementação e planos de projeto. É responsabilidade do solucionador entender as potencias fontes de restrições para cada ambiente específico da aplicação e determinar o impacto de cada restrição sobre o potencial espaço da solução.

Tabela 4–4 Potenciais restrições do sistema

Fonte	Exemplo de Considerações
Econômica	▪ Que restrições financeiras ou orçamentárias são aplicáveis? ▪ Existem custos associados nas vendas de mercadorias ou considerações sobre preço de produtos? ▪ Existe algum problema de licenciamento?
Política	▪ Existem problemas políticos internos ou externos que possam, potencialmente, afetar a solução? ▪ Existem problemas interdepartamentais?
Técnica	▪ Temos restrições quanto à escolha de tecnologia? ▪ Temos restrições para trabalhar com a plataforma ou tecnologias existentes? ▪ Utilizaremos algum pacote de software adquirido?
Sistêmica	▪ A solução será construída sobre o sistema existente? ▪ Devemos manter compatibilidade com a solução existente? ▪ Que sistemas operacionais e ambientes devem ser suportados?
Ambiental	▪ Existem restrições ambientais ou legais? ▪ Existem requisitos de segurança? ▪ Estamos restritos a algum padrão?
De planejamento e recursos	▪ O planejamento está definido? ▪ Estamos restritos aos recursos existentes? ▪ Podemos utilizar trabalho externo? ▪ Podemos aumentar os recursos temporários ou permanentes?

Retornando ao nosso exemplo, quais restrições podem ser impostas sobre o novo sistema de pedidos? A Tabela 4–5 sumariza os recursos e restrições que foram impostas sobre o novo sistema.

Tabela 4–5 Restrições, fontes e lógica para o sistema de pedidos

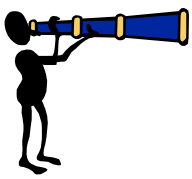
Fonte	Restrição	Lógica
Operacional	Uma cópia exata dos dados dos pedidos de venda deve permanecer na base de dados legado por um ano.	O risco de perda de dados é muito grande; nós precisamos executar em paralelo durante um ano.
Sistema e SO	A aplicação não deve utilizar mais que 20 megabytes da memória RAM do servidor.	Existe limitação de memória no servidor.
Orçamento de equipamentos	O sistema deve ser desenvolvido no servidor existente; novos hardwares clientes para usuários serão fornecidos.	Controle de custos e conservação do sistema existente.
Orçamento de pessoal	Recursos fixos de pessoal; sem terceirização.	Custos operacionais fixos de acordo com o atual orçamento.
Tecnologia obrigatória	Será utilizada a nova metodologia orientada a objetos.	Nós acreditamos que esta tecnologia irá aumentar a produtividade e elevar a confiabilidade do software.

Sumário

Completado este passo, podemos ficar razoavelmente confiantes de que conseguimos:

- Entender o problema a ser resolvido, bem como as causas raízes do problema.
- Identificar os stakeholders que, com seu julgamento coletivo, irá, ao final, determinar o sucesso ou o fracasso do nosso sistema.
- Obter uma noção da fronteira da solução.
- Conhecer as restrições e o grau de liberdade que temos para solucionar o problema.

Vislumbrando o Futuro



Com esta base conceitual, podemos agora voltar nossa atenção para duas técnicas mais específicas de solucionar problemas, as quais podem ser aplicadas em certos domínios de aplicações. No Capítulo 5, nós veremos a *modelagem de negócio*, uma técnica que podemos aplicar em aplicações de Sistemas de Informação - IS/IT (*Information system / information technology*). No Capítulo 6, veremos a *Engenharia de Sistemas* para sistemas de software intensivo, que pode ser aplicado para aplicação do domínio de sistemas embutidos.

Quanto ao terceiro domínio - ISV (*Independent software vendors*), pertencente a fornecedores independentes de software, as técnicas de análise de problemas estão normalmente focadas nas seguintes atividades:

- Identificar as oportunidades e segmentos de mercado.
- Identificar as classes de potenciais usuários e suas necessidades particulares.
- Estudar a demografia da potencial base de usuários.
- Entender o potencial da demanda, do preço e da flexibilidade de preços.
- Entender as estratégias de venda e canais de distribuição.

Claramente, estes tópicos são interessantes, mas para nos ajudar a gerenciar o escopo deste volume, nós não iremos explorar tais assuntos específicos. No entanto, você pode ficar confiante e seguro de que as habilidades de equipe que iremos explorar nos próximos capítulos irão se aplicar igualmente bem para esta classe de aplicação, como iremos demonstrar.

Nota: Uma das coisas mais difíceis que encontramos ao escrever este livro foi tentar apresentar várias técnicas para construir o conjunto de habilidades de equipe. Nenhuma técnica funciona em todas as situações; nunca duas situações serão as mesmas.

Nos capítulos anteriores, focamos numa abordagem geral e filosófica da análise de problemas que parece funcionar em vários contextos de sistemas. No entanto, esse problema de "seleção de técnicas para aplicar" se transformará em algo muito mais sério nos próximos capítulos, onde definiremos a técnica de modelagem de negócio, a técnica de engenharia de sistemas, e continuamos a definir uma variedade de técnicas na Habilidade de Equipe 2, Entendendo as Necessidades do Usuário, onde apresentaremos uma grande variedade de técnicas que podem ser usadas para entender as necessidades dos stakeholders e de usuários com respeito ao "*sistema que você irá construir*".

No entanto, nós pensamos que seja importante destacar que as técnicas descritas neste livro – da análise do problema até brainstorming – podem ser usadas em diferentes partes do processo de software e não apenas na parte do processo onde nós escolhemos descrevê-las. Por exemplo, a equipe pode usar a análise de problemas para definir um problema de sistema de pedidos ou resolver um problema técnico dentro de sua implementação. Da mesma forma, a equipe pode usar o brainstorming para determinar as prováveis causas raízes num exercício de análise de problemas ou determinar as prováveis características novas de um sistema como fizemos no capítulo 5. Não faremos nenhuma tentativa para descrever todas as circunstâncias sob as quais uma particular técnica poderá ser aplicada, mas, ao invés disso, nos concentraremos em fazer com que a equipe desenvolva habilidades para que possa aplicar essas técnicas e incluí-las em sua maleta de dicas e truques – para pegar e usar em momentos apropriados do projeto.

Capítulo 5

Modelagem de Negócio

Pontos chaves

- A modelagem de negócio é uma técnica de análise de problema apropriado especialmente para ambientes de IS/IT.
- A modelagem de negócio é usada para ajudar a definir sistemas e suas aplicações.
- Um modelo use case de negócio, que consiste de atores e use cases, é um modelo das funções pretendidas do negócio.
- Um modelo de objetos de negócio descreve as entidades que fornecem a funcionalidade para realizar os use cases de negócio, e como essas entidades se interagem.

No contexto de um ambiente de tecnologia de sistemas informação (IS/IT), o primeiro problema a ser resolvido é a sua vasta abrangência como descrevemos no Capítulo 4. Neste ambiente, a complexidade de negócio é abundante, e normalmente precisamos entender algo sobre essa complexidade antes de tentar definir um problema específico importante a ser resolvido. Esse ambiente possui não apenas de um usuário ou dois com suas interfaces computacionais, mas de organizações, unidades de negócio, departamentos, funções, ampla rede de computadores, intranet e extranet corporativa, clientes, usuários, recursos humanos, sistemas MRP (Material Requirement Planning), estoque, sistemas de gerenciamento, entre outros.

Além disso, mesmo que estejamos concentrados numa aplicação específica que será implementada, devemos constantemente nos lembrar do contexto mais amplo no qual essa aplicação estará inserida. Talvez isso possa ser realizado com maior sucesso fazendo-se questões corretas, mas como qualquer técnica, existem mais coisas que podem ser feitas num contexto específico do que nos casos genéricos.

No contexto de IS/IT, poderia ser útil ter uma técnica que possibilite determinar as respostas para as seguintes questões:

- Por que construir um sistema dessa maneira?
- Onde deve ser alocado?
- Como podemos determinar quais funcionalidades devem ser alocadas num particular sistema?
- Quando devemos utilizar o processamento manual ou soluções de contorno?
- Quando devemos considerar a reestruturação da organização a fim de solucionar o problema?

Felizmente, existe uma técnica que se encaixa perfeitamente para resolver este problema, e esta técnica é a *modelagem de negócio*.

Propósito da Modelagem de Negócio

No contexto deste volume, podemos pensar nos termos "negócio" e "modelagem de negócio" como algo tão amplo quanto possível. Por exemplo, o nosso negócio pode ser o negócio de desenvolvimento de software ou de fabricação de robôs soldadores, ou você pode querer modelar um negócio de filantropia, organizações de serviço, processo intradepartamental ou fluxo de trabalho interno.

De qualquer forma, o propósito da modelagem de negócio possui duas partes:

- Entender a estrutura dinâmica da organização
- Assegurar que clientes, usuários finais e desenvolvedores tenham um entendimento comum da organização.

Essa abordagem dá à equipe uma maneira lógica de definir onde a utilização do software pode melhorar a produtividade do negócio e ajudar na determinação de requisitos para essa utilização.

Usando Técnicas de Engenharia de Software para Modelar Negócios

Naturalmente, várias técnicas podem ser aplicadas para se modelar negócios. No entanto, seria conveniente que, como desenvolvedores de software, tivéssemos à nossa disposição um conjunto rico em ferramentas e técnicas já usadas na modelagem de nossos softwares. De fato, nós já sabemos modelar entidades (objetos e classes), relacionamentos (dependências, associações, entre outros), processos complexos (seqüência de atividades, transições de estados, eventos, condicionais, entre outros) e outros construtores que ocorrem naturalmente no contexto de nossos projetos de aplicações de software.

Com a escolha da técnica correta para modelar negócios, modelos use cases e modelos de objetos, se tornarão artefatos comuns durante a atividade de solucionar problemas.

Se nós pudemos aplicar essas mesmas técnicas para modelar negócios, poderíamos falar a mesma linguagem em ambos os contextos. Por exemplo, uma "coisa", tal como o contracheque da folha de pagamento, descrita no domínio de negócio, pode relacionar-se com uma "coisa" que aparece novamente no domínio de software – por exemplo, o registro do contracheque da folha de pagamento. Se nós tivermos sorte o suficiente para usar as mesmas técnicas, ou técnicas muito similares, tanto para a análise de problemas quanto para o projeto de software, ambas poderiam se beneficiar compartilhando os resultados produzidos pelas duas atividades.

Escolhendo a Técnica Correta

Historicamente, vimos que as técnicas de modelagem que foram desenvolvidas e maturadas no domínio do software inspiraram novas maneiras de visualizar uma organização. Desde que as técnicas de modelagem visual se tornaram comuns em projetos de novos softwares, usar técnicas similares no domínio de negócio tornou-se natural. Essa metodologia foi bem desenvolvida por Jacobson, Ericsson e outros em 1994.

Nos anos 80 e 90, houve uma rápida proliferação tanto das técnicas de modelagem de negócio quanto de metodologias de desenvolvimento de software. No entanto, eles são todas diferentes! No centro dessas atividades estavam os vários métodos e notações orientadas a objetos desenvolvidos por diversos especialistas em engenharia de software e pesquisadores. Felizmente, a guerra de metodologias foi superada, e a indústria assentou-

se sobre um padrão industrial – a UML (Unified Modeling Language) – para a modelagem de sistemas de software intensivos.

A Linguagem de Modelagem Unificada

No final de 1997, uma linguagem gráfica para visualizar, especificar, construir e documentar os artefatos de um sistema intensivo de software foi adotado como um padrão industrial (Booch, Jacobson e Rumbaugh). A UML fornece um conjunto de elementos de modelagem, notações, relacionamentos e regras de uso que podem ser aplicados na atividade de desenvolvimento de software. No entanto, a UML pode também ser aplicada para modelar sistemas e negócios. Um tutorial sobre UML está fora do escopo deste volume. (Para isso, consulte o livro dos três amigos sobre a UML: Booch, Rumbaugh e Jacobson (1990), *The Unified Modeling Language, User Guide*; e Rumbaugh, Booch e Jacobson (1998), *The Unified Modeling Language, Refrence Manual*). No entanto, iremos usar alguns conceitos chaves da UML nesta seção e construir outros a partir desses conceitos nas seções subseqüentes.

Modelagem de Negócio Usando UML

Uma das metas da modelagem de negócio é desenvolver um modelo de negócio que possa ser usado para orientar o desenvolvimento de aplicações. Os dois construtores chaves de modelagem que pode ser usado para este propósito são: *modelo use-case de negócio* e *modelo de objetos de negocio*.

Um *modelo use-case de negócio* é um modelo das funcionalidades pretendidas do negócio e é usado como uma das fontes essenciais para identificar papéis e produtos da organização. Como tal, o modelo use-case de negócio consiste de atores de negócio – usuários e sistemas que interagem com o negócio – e os use cases de negócio – seqüência de eventos pelos quais os atores de negócio interagem com os elementos de negócio para realizar seu trabalho. Juntos, atores de negócio e use cases de negócio descrevem quem está envolvido nas atividades de negócio e como essas atividades ocorrem. A Figura 5– 1 ilustra o modelo use-case de negócio. Note que os ícones possuem uma barra, indicando que o ator e o use case são de negócio ao invés de serem de sistema.

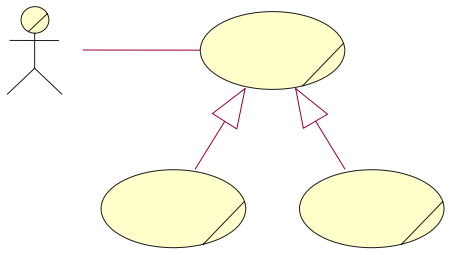


Figura 5–1 Modelo use-case de negócio

Um modelo use-case de negócio, então, consiste de atores de negócio e use case de negócio, com atores de negócio representando papéis externos ao negócio (por exemplo, clientes e fornecedores) e use cases de negócio representando processos. Abaixo estão alguns exemplos de use cases de negócio:

- "Liberar pagamento de empregados"
- "Negociar os termos do contrato com o cliente"

Exemplos de atores de negócio:

- "Cliente"
- "Fornecedor"
- "Secretaria dos Negócios da Fazenda do Estado"

O *modelo de objetos de negócio* descreve as entidades – departamentos, contadores, sistemas – e como eles se interagem para gerar as funcionalidades necessárias a fim de realizar os use cases de negócio. A Figura 5–2 representa o modelo de objetos de negócio. Os ícones, com um ator dentro de uma circunferência, representam um worker (trabalhador) que aparece dentro do processo de negócio, tais como secretária, chefe ou diretor. As circunferências com as barras representam uma entidade de negócio ou alguma coisa que os workers de negócio usa ou produz, tais como contratos, cadastro de funcionários e produtos.

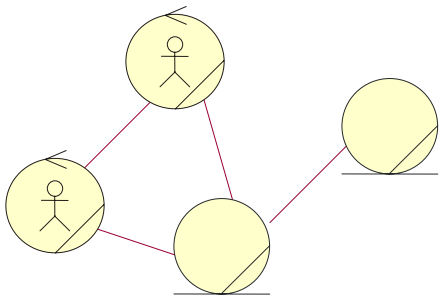


Figura 5–2 Modelo de objetos de negócio

Um modelo de objetos de negócio também possui realizações de use-cases de negócio, os quais mostram como os use cases de negócio são "executados" em termos de interações de workers de negócio e entidades de negócio. Para refletir grupos de departamentos numa organização, workers de negócio e entidades de negócio podem ser agrupados dentro de unidades organizacionais.

Todos juntos, os dois modelos fornecem uma visão geral compreensiva de como o negócio funciona e permite que a equipe de desenvolvimento concentre-se nas áreas no qual o sistema pode ser fornecido com o objetivo de elevar a eficiência geral do negócio. Os modelos também ajudam a equipe a entender quais mudanças terão que ser realizadas dentro do processo de negócio a fim de que o novo sistema possa ser efetivamente implementado.

Da Modelagem de Negócio aos Modelos de Sistemas

Uma das vantagens desta abordagem de modelagem de negócio é a maneira clara e concisa de mostrar dependências entre modelos do negócio e modelos do sistema. Esta claridade eleva a produtividades do processo de desenvolvimento de software e também ajuda a assegurar que o sistema que está sendo desenvolvido resolve as reais necessidades do negócio. Veja a figura 5–3.

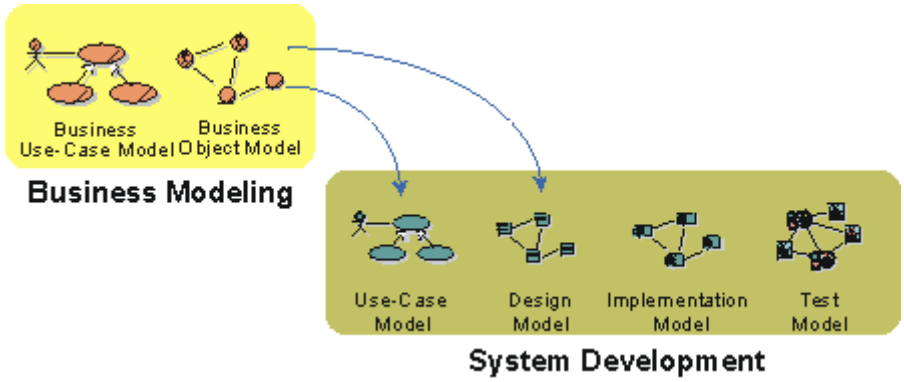


Figura 5–3 Modelos de negócio/sistema

A transição entre os dois modelos pode ser resumida da seguinte forma:

- Workers de negócio irão se tornar atores do sistema que nós desenvolvemos ou um elemento do próprio sistema. Por exemplo, caixa de um banco e caixa eletrônico de um banco.
- Comportamentos descritos para os workers de negócio são coisas que podem ser automatizados, de tal forma que eles irão ajudar a encontrar os use cases de sistema e definir as funcionalidades necessárias.
- Entidades de negócio são coisas que queremos que o sistema nos ajude a manter. Assim, eles podem ajudar a encontrar classes entidades do modelo de análise do sistema.

Ao realizar a transição, a modelagem de negócio facilita o processo de ir do entendimento do negócio e do problema dentro do negócio para potenciais aplicações que podem ser implementadas para gerar as soluções do problema identificado.

Quando Usar a Modelagem de Negócio

A modelagem de negócio não é algo que recomendamos para todos os projetos de engenharia de software. Modelos de negócio adicionam valor quando o ambiente da aplicação é complexa e multidimensional, e quando muitas pessoas estão diretamente envolvidas no uso do sistema. Por exemplo, se você estiver construindo uma característica adicional para um componente de chaveamento de telecomunicações, você não deveria usar a modelagem de negócio. Por outro lado, se você estiver construindo um sistema de pedidos para o GoodsAreUs, nós

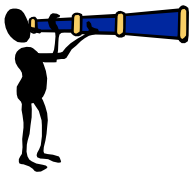
recomendamos que utilize a modelagem de negócio para ganhar vantagem na análise do problema.

Sumário

Neste capítulo, nós descrevemos uma técnica específica de análise de problema, a modelagem de negócio. Ao fazer isso, definimos:

- Porque você precisa modelar o negócio.
- Como, usando a UML, nós transpomos as técnicas de desenvolvimento da engenharia de software e a usamos para a modelagem de negócio.
- Os principais artefatos da modelagem de negócio, o modelo use-case de negócio e o modelo de objetos de negócio.
- Como você pode definir aplicações de software e gerar requisitos de software a partir dos modelos de negócio.

Vislumbrando o Futuro



No próximo capítulo, veremos a engenharia de sistemas para sistemas de software, uma outra técnica de análise de problemas, a qual irá ajudar a formatar a aplicação de sistemas do tipo embutido.

Capítulo 6

Engenharia de Sistemas de Software Intensivos

Pontos chaves

- A engenharia de sistemas é uma técnica de análise de problemas especialmente apropriada para desenvolvimento de sistemas embutidos.
- A engenharia de sistemas nos ajuda a entender requisitos impostos à aplicações de software que executam dentro da solução sistêmica.
- O flowdown[1] de requisitos é antes de tudo, uma questão de assegurar que todos os requisitos de sistema estarão cobertos por um subsistema ou pela colaboração de um conjunto de subsistemas.
- Atualmente, o sistema deve ser otimizado para reduzir custos de software ao invés de reduzir custos de hardware.

No Capítulo 5, vimos a modelagem de negócio, uma técnica de análise de problemas para aplicações de sistemas de informação (IS/IT). A modelagem de negócios nos ajuda a determinar quais aplicações devem ser construídas e onde devem ser executadas, dentro do ambiente computacional da empresa e departamentos, edifícios, e construções políticas e físicas da empresa. Em outras palavras, essa análise pode nos ajudar a determinar *por que* e *onde* uma aplicação deverá existir. Ao fazer isso, naturalmente, fizemos uma sutil mudança do *espaço do problema* para uma visão inicial do *espaço da solução*, onde a funcionalidade que resolve o problema irá existir em uma ou mais aplicações que atendem as necessidades finais do usuário.

No negócio de sistemas embutidos, no entanto, o domínio do problema e o domínio da solução parecem totalmente diferentes. Ao invés de departamentos, pessoas e processos, o domínio consiste de conectores, fontes de energia, racks de equipamentos, componentes eletrônicos e elétricos, periféricos de controle de fluidos, outros sistemas de software, subsistemas mecânicos e ópticos, entre outros. Aqui, a modelagem de negócio não pode ajudar muito. Ao invés disso, devemos adotar uma estratégia diferente para responder o *por que* e *onde* a aplicação deve existir.

O que é Engenharia de Sistemas?

De acordo com o conselho internacional de engenharia de sistemas (INCOSE - International Council on Engineering Systems, 1999):

> *A engenharia de sistemas é uma abordagem interdisciplinar e pretende viabilizar a realização de sistemas de sucesso. O foco está em definir as*

[1] É o processo de derivar e alocar requisitos a todos os níveis da decomposição sistêmica.

necessidades dos clientes e funcionalidades solicitadas no início do ciclo de desenvolvimento, documentando requisitos e então, procedendo com a síntese do projeto e validação do sistema considerando, como um todo, o problema de:

- *Operação*
- *Desempenho*
- *Teste*
- *Manufatura*
- *Custo e Cronograma*
- *Treinamento e suporte*
- *Disponibilização.*

A engenharia de sistemas integra todas as disciplinas e grupos de especialidades dentro de um esforço de equipe, formando assim, um processo de desenvolvimento estruturado que progride do conceito, para a produção e chegando na sua operação. A engenharia de sistemas considera tanto o negócio quanto as necessidades técnicas de todos os clientes com o objetivo de fornecer um produto de qualidade que atenda as necessidades dos usuários.

Ufa! Essa definição é muito longa! No entanto, pela definição, a engenharia de sistemas pode ser considerada uma técnica de análise de problemas, embora que, neste volume, não esperamos cobri-la em sua totalidade. Para maiores detalhes, veja Rechtin (1997).

No escopo deste volume, a engenharia de sistemas pode nos ajudar a entender as necessidades do espaço do problema e os requisitos que são impostos à solução. Nesse contexto, a engenharia de sistemas nos ajuda a entender os requisitos que são impostos às aplicações de software e que executem dentro da solução sistêmica. Em outras palavras, nós aplicamos a engenharia de sistemas como uma técnica de análise de problemas para nos ajudar a entender os requisitos de nossas aplicações de software, se elas executarem sobre um microprocessador embutido ou num sistema UNIX dentro do contexto de um sistema mundial de telecomunicação.

Princípios Pragmáticos da Engenharia de Sistemas

Se considerarmos que a engenharia de sistemas é uma técnica de análise de problemas, os passos específicos, ou pelo menos os princípios básicos da disciplina, devem nos fornecer os passos necessários para aplicar a engenharia de sistemas para analisar o problema dentro do nosso contexto de requisitos. O INCOSE definiu um conjunto básico de 8 princípios de engenharia de sistemas (1993):

- Conhecer o problema, conhecer o cliente e conhecer o consumidor.
- Usar critérios efetivos com base nas necessidades para tomar decisões sistêmicas.
- Estabelecer e gerenciar requisitos.
- Identificar e avaliar alternativas de forma a convergir para um solução.
- Verificar e validar requisitos e performance da solução.
- Manter a integridade do sistema.
- Usar um processo articulado e documentado.
- Gerenciar frente a um plano.

Esta lista identifica alguns princípios pragmáticos da engenharia de sistemas. No entanto, a verdade é que um subconjunto da disciplina de engenharia de sistemas fundamenta-se num outro processo, o da decomposição sucessiva de sistemas complexos em sistemas mais simples.

A Composição e Decomposição de Sistemas Complexos

Com este processo, um problema complexo, o sistema (Figura 6–1), é decomposto em problemas menores – subsistemas (Figura 6–2). Cada subsistema pode ser pensado, projetado e manufaturado com sucesso, e então integrado para produzir a solução sistêmica.

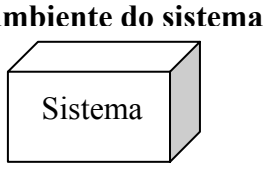


Figura 6–1 Um sistema em seu ambiente

A disciplina de engenharia que suporta a abordagem da decomposição sistêmica utiliza atributos da definição anterior, tal como o entendimento da característica operacional, manufatura, teste, entre outras.

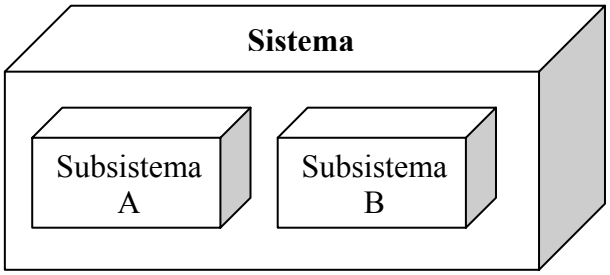


Figura 6–2 Um sistema composto por dois subsistemas

Este processo de decomposição, ou refinamento sucessivo, prossegue até que o engenheiro de sistemas atinja os resultados *corretos*, de acordo com as medidas quantitativas específicas do domínio da engenharia de sistemas. E muitos casos, os subsistemas definidos na composição inicial são, por sua vez, decompostos em outros subsistemas, como ilustra a Figura 6–3.

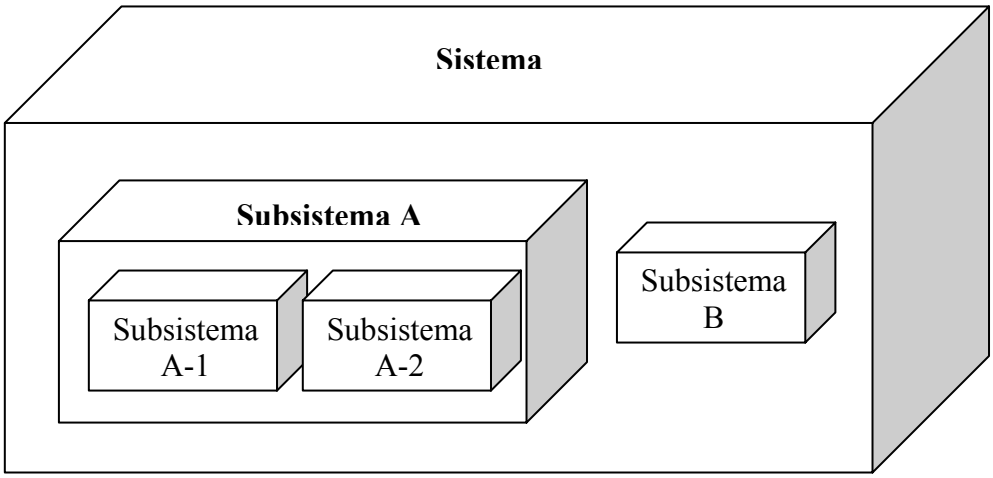


Figura 6–3 Um subsistema composto por dois subsistemas

Em muitos sistemas complexos, esse processo continua até que um grande número de subsistemas tenha sido desenvolvido. Dizem que a aeronave F22, por exemplo, é composta de 152 subsistemas.

O engenheiro de sistemas descobre que o trabalho realizado está correto quando:

- A distribuição e particionamento das funcionalidades estiverem otimizadas para atingir a funcionalidade global do sistema com o mínimo de custo e máxima flexibilidade.
- Cada subsistema puder ser definido, projetado e construído por uma equipe de tamanho pequeno, ou ao menos modesto.
- Cada subsistema puder ser manufaturado dentro das restrições físicas e tecnológicas do processo de manufatura disponível.
- Cada subsistema puder ser seguramente testado como um subsistema, com disponibilidade de acessórios e peças apropriadas que simulem as interfaces com outros sistemas.
- Considerações apropriadas do domínio físico – o tamanho, peso, localização, e distribuição dos subsistemas – tiverem sido otimizados no contexto global do sistema.

Alocação de Requisitos na Engenharia de Sistemas

O processo flow-down de requisitos aloca funcionalidades do sistema aos subsistemas.

Mesmo que a engenharia de sistemas tenha propiciado realizar um bom trabalho de definir os requisitos do sistema, o problema de gerenciamento de requisitos ainda não está resolvido. O que se passa com esses subsistemas? Quais requisitos são impostos para cada subsistema? Em muitos casos, o processo é o de associar os requisitos do nível de sistema aos subsistemas ("Subsistema B irá executar o algoritmo de velocidade instantânea e monitorar diretamente o display de alerta"). Este processo distribuir requisitos (flow-down) é o principal meio de assegurar que todos os requisitos do sistema serão atendidos por um subsistema em algum lugar ou por um conjunto de subsistemas que se colaboram.

Sobre Requisitos Derivados

Algumas vezes, descobrimos que nós criamos uma nova classe de requisitos – *requisitos derivados* – que podem ser impostos aos subsistemas. Tipicamente, existem duas subclasses de requisitos derivados.

1. Requisitos de subsistemas são aqueles que devem ser impostos apenas a subsistemas, mas não necessariamente fornece um benefício direto ao usuário final ("Subsistema A deve executar o algoritmo que computa a velocidade instantânea da aeronave").

2. Requisitos de interface podem aparecer quando os subsistemas necessitarem se comunicar uns com os outros para atingir um resultado global. Eles precisarão compartilhar dados, fonte de energia, ou um algoritmo computacional de uso comum. Nesses casos, a criação de subsistemas também propicia a criação de interfaces entre subsistemas (veja a Figura 6–4).

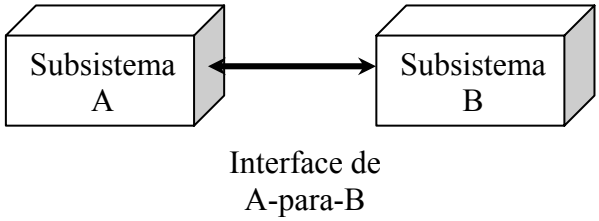


Figura 6–4 Interface entre dois subsistemas

Mas existem mesmo requisitos derivados? Nós tratamos os requisitos derivados como qualquer outro requisito? Eles não parecem atender as definições do Capítulo 2 (embora possa atenda definições sobre restrições de projeto que iremos falar mais adiante).

É importante reconhecer que esses requisitos, embora cruciais para o sucesso do projeto, são *derivados* do processo de decomposição do sistema. Dessa forma, decomposições alternativas criam requisitos alternativos, e esses requisitos não são cidadãos de primeira classe no sentido de que eles não refletem requisitos originados de nossos clientes. No entanto, a partir do ponto de vista de um fornecedor de um subsistema, eles são cidadãos de primeira classes porque refletem requisitos impostos pelo cliente (o desenvolvedor do sistema).

Não existe resposta mágica. O como tratar esses requisitos é baseado no papel da equipe de desenvolvimento no projeto, na decomposição de sistemas, bem como em outros fatores tecnológicos. Assim, é importante saber "como chegar aqui" e tratar os requisitos de forma apropriada. É importante reconhecer que a especificação de requisitos derivados irá, no final, afetar a habilidade do sistema de fazer seu trabalho, bem como a manutenibilidade e robustez do sistema.

Uma Revolução Silenciosa

Na indústria, a inteligência, antes existente nos componentes de hardware, foram alocadas aos componentes de software.

A engenharia de sistemas tem sido, tradicionalmente, uma disciplina aplicada principalmente a sistemas físicos, tais como sistemas de aeronaves, sistemas de freios de automóveis, sistemas de fontes de energia e aparelhos de consumo de energia, entre outros. No entanto, durante os últimos 20 anos, uma revolução silenciosa vem ocorrendo na engenharia de sistemas de sistemas complexos. Gradualmente, nas indústrias de transportes, telecomunicações, equipamentos industriais, equipamentos médicos, instrumentos científicos, e em muitas outras indústrias, os sistemas e equipamentos vêm se tornando cada vez menores. Para atender a elevação da complexidade e sofisticação, mais e mais funcionalidades estão sendo alocadas aos subsistemas de software ao invés de serem alocadas aos componentes de hardware. Afinal de contas, o software é flexível; muitos algoritmos de medição, análise e detecção são mais fáceis, ao menos muito mais baratos, de serem implementados em software do que em hardware. O mais importante é que são muito mais fáceis de serem mudados.

Assim, na indústria, a inteligência, antes existente nos componentes de hardware, viabilizada através da combinação de sistemas elétricos e eletrônicos, sistemas mecânicos, e sistemas químico-físicos; encontram-se hoje, implementadas em componentes de software, imersos dentro de microprocessadores ou subsistemas.

Quando as Gerações se Colidem: os Anciões Encontram os Jovens Arrogantes

Por décadas, engenheiros de sistemas ocuparam, na indústria, os principais cargos de engenheiro de projeto sênior. Marcado por ferimentos e testados em campo, vários desses *engenheiros de sistemas seniores* eram especialistas em disciplinas específicas, tais como: engenharia mecânica e eletrônica; e muitos eram alguns dos melhores generalistas da equipe. Eles eram testemunhas de grandes desastres e haviam conquistado muitas vitórias. Agora, velhos e experientes, eles possuem, incrivelmente bem, o conhecimento específico do domínio da aplicação – rádios, aeronaves, refrigeração, robótica, equipamentos de controle de materiais – e sabem também diferenciar facetas técnicas, econômicas e políticas envolvidas na tecnologia de implementação.

Mas, de repente, uma nova geração de indivíduos invadiu a sua área. Esses recém-chegados – os *programadores*, ou nos bons dias, engenheiros de software – relativamente inexperientes em sistemas complexos, não sabiam diferenciar o peso, da balança, ou a otimização sistêmica de seu próprio umbigo; mas eles podiam fazer um microprocessador cantar em linguagem assembly. Além disso, pareciam que eram formados por genes diferentes ou, pelo menos, por uma geração diferente, o qual adicionava complexidade do conflito cultural e de gerações ao processo de engenharia de sistemas. Muitas situações interessantes ocorreram.

Por algum tempo, a batalha estava equilibrada: engenheiros de sistemas atendiam as solicitações de particionar e alocar funcionalidades finais para o sistema. Mas em muitas indústrias, a tecnologia de software assumiu gradualmente o controle, e a engenharia de sistemas havia sido dominado, pelo menos em parte, devido à necessidade de atribuir ao sistema, funcionalidades flexíveis de software. Existem inúmeras razões técnicas e sólidas para essa transição. Com o tempo, vários fatos ficaram óbvios:

- É o software, não o hardware, que irá determinar as funcionalidades finais do sistema, bem como o sucesso do sistema nas mãos de usuários e no mercado.
- É o software, não o hardware, que irá consumir a maior parte dos custos de pesquisa e desenvolvimento do sistema.
- É o software, não o hardware, que estará no caminho crítico e irá, dessa forma, determinar finalmente quando o sistema irá para o mercado.
- É o software, não o hardware, que irá absorver a maioria das mudanças ocorridas durante o desenvolvimento de sistemas e que será evoluído, no passar dos anos, para atender as necessidades por mudanças do sistema.

E, talvez o mais surpreendente:

- Os custos do desenvolvimento e manutenção de software definem os custos agregados e amortizados durante o ciclo de vida do produto, tornando-se o principal componente na formação dos preços de venda dos produtos, em alguns casos igual ou maior que o do hardware, tal que se tornou o santo gral dos fabricantes de sistemas: *custo total de fabricação.*

Agora, muitos sistemas devem sofrer otimizações de software, não de hardware.

Este último fato deu o golpe de misericórdia, pois indica que você deve considerar para otimização dos custos do sistema, não o hardware ou manufatura, mas o *desenvolvimento, manutenção, evolução e aprimoramento de software contidos no sistema*. Isso mudou significativamente o jogo. Por agora, a engenharia de sistemas deve ser realizada considerando o computador a ser utilizado. Isso significa que a engenharia de sistemas deve, normalmente:

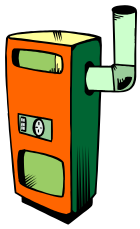
- Maximizar a habilidade de executar software, fornecendo mais do que recursos adequados de computação, mas adicionando mais microprocessadores, RAM, ROM, memória de armazenamento de massa, largura de banda, ou quaisquer recursos que o sistema necessite para executar seu software e ajudar a reduzir o custo de venda de produtos.
- Fornecer interfaces de comunicação adequadas entre subsistemas e assegurar que o mecanismo de comunicação escolhido (Ethernet, Firewire, porta serial, ou single data line) seja extensível, via mecanismos de software e não de hardware.

Um após o outro, essas mudanças afetaram os desafios do gerenciamento de requisitos de duas maneiras:

- Cada uma dessas dimensões irá criar novos *requisitos* que o sistema hardware deverá preencher a fim de satisfazer a solução a ser construída.
- A maior parte do problema de requisitos se deslocou para a aplicação de software.

Felizmente, para menos para a segunda maneira, este é o objetivo deste volume, e nós esperamos prepará-lo muito bem para este particular problema.

Evitando o problema do sistema de chaminés



Isso tudo é bom, ao menos, na maioria das vezes. Lidar com sistemas complexos requer abordagens não-triviais, e o sistema de subsistemas é um meio para este fim. Certamente, as alternativas são piores, nós poderíamos chegar ao final com um sistema incrivelmente complexo e que ninguém conseguisse entender, com comportamentos indeterminados e projeto baseado em funcionalidades compartilhadas, particionamento pobre, e códigos concorrentes que nunca serão desatados. A engenharia de sistemas parece ser a melhor alternativa.

Como isso afeta o gerenciamento de requisitos? Quando é feita a contagem final, descobrimos que os requisitos de subsistemas são muito mais numerosos do que os requisitos externos, ou daqueles que afetam o comportamento do sistema no ambiente do usuário. No fim, nós investimos mais na priorização, gerenciamento, e identificação de requisitos de subsistemas do que naqueles que afetam o usuário final. Isso não parece ser uma coisa completamente positiva.

E o que acontece se nós não fizermos um bom trabalho de engenharia de sistemas? O sistema se tornará frágil e resistirá a mudanças porque o peso das propriedades de requisitos irá "amarrar" nossa implementação. Os nossos requisitos de subsistemas tomarão o controle da flexibilidade de projeto, e uma mudança num subsistema afetará outros subsistemas. É esse o sistema de "chaminés" da legenda, e como tal, resistente a mudanças. Em interfaces de subsistemas, o problema pode ser pior. Se as interfaces não forem especificadas apropriadamente, o sistema se tornará frágil e não estará apto a evoluir para atender as necessidades de mudanças; a menos que sejam feitas substanciais alterações nas interfaces e em todos os subsistemas dos quais dependem dessas interfaces.

Quando Subsistemas São Subcontratos

Surge mais uma outra complicação. Uma vez que subsistemas são, normalmente, desenvolvidos por diferentes equipes – a final de contas, esta é uma das razões de criarmos subsistemas – os requisitos de subsistemas e interfaces tendem, por sua vez, a se tornarem *contratos* entre as equipes. ("Meu subsistema apresenta os resultados da computação de velocidade instantânea neste exato formato..."). De fato, em alguns casos, um subsistema pode ser desenvolvido por um subcontratado, cuja nota fiscal possui um logotipo diferente do seu. Neste caso, o nosso desafio de requisitos deixa o contexto técnico e sistêmico e passa a ser a da política do "futebol". ("Darn. Os requisitos não podem ser mudados a menos que o contrato seja renegociado"). Em breve, o projeto pode estar intimamente "amarrado à suas calças". Uma forte advertência: *Muitas tentativas de se construir grandes sistemas encontraram sua morte nas mãos deste problema*.

Fazendo o Trabalho de Corretamente

O que devemos fazer? Bem, fazer um bom trabalho de engenharia de sistemas é o objetivo principal. Como participante dessa atividade de desenvolver sistemas de software intensivos, você precisará considerar as seguintes recomendações:

- Desenvolver, entender e manter os requisitos e use cases que permeiam os subsistemas e que descrevem a funcionalidade global do sistema em alto nível. Tais use cases fornecem o contexto de como o sistema deverá trabalhar e

assegurar que você "não se perdeu na floresta entre as árvores" Eles também ajudarão a assegurar que a arquitetura do sistema foi projetada para sustentar os principais cenários.

- Faça o melhor trabalho possível de particionar e isolar funcionalidades de subsistemas. Use princípios da tecnologia de objetos: encapsulamento e ocultamento de informações, interface por contrato, mensagens ao invés de compartilhamento de dados – em seu trabalho de engenharia de sistemas.
- Se possível, desenvolva o software como um todo, não como um monte de peças individuais, um para cada subsistema físico. Uma das características do sistema de chaminés é que em ambos os lados da interface (bem ou mau definidas), o software precisa reconstruir o estado dos elementos chaves (objetos) que são necessários para tomar decisões em ambos os lados; diferentemente do hardware, cuja alocação dos requisitos em ambos os lados não representa uma divisão clara.
- Quando codificar interfaces, use códigos comuns em ambos os lados da interface. Caso contrário, surgirão variações sutis, freqüentemente provocados pelas "otimizações", isso fará com que a sincronização de estados torne-se muito difícil. Assim, se a fronteira entre os dois subsistemas físicos posteriormente desaparecerem – isto é, se a engenharia de sistemas achar que os processadores são suficientes para suportar ambos os subsistemas A e B – o engenheiro de software irá ter um momento difícil para consolidar as duas peças de software (se forem diferentes, é claro!).
- Definir especificações de interface que possam fazer mais do que o necessário, ou seja, mais do que simplesmente atender as condições conhecidas. Investir numa largura de banda um pouco maior, portas de entrada/saída extras, ou adicionar alguns slots a mais para permitir futuras expansões.

Finalmente, veja se você pode encontrar alguns daqueles anciões para lhe ajudar com a engenharia de sistemas. Eles estiveram aqui antes, e sua experiência pode ser muito valiosa para você. Além disso, você poderia assim, a ajudar a por um fim nesse conflito de gerações!

Uma estória: Particionando Grandes Sistemas de Software em Subsistemas para Equipes Distribuídas de Desenvolvimento

Numa aula, Rusty, um gerente de software experiente, nos abordou e definiu o seguinte problema, descrito como um diálogo:

Rusty: E*stamos construindo uma grande aplicação que será executada num único sistema servidor. Nosso possuímos duas equipes separadas, contendo cada uma 30 pessoas; uma das equipes vive no lado leste do rio da cidade de Nova York, e o outro vive no lado oeste. As duas equipes possuem gerentes diferentes e competências diferentes. Como dividir o trabalho e criar um sistema que funcione?*

Nós: *Bem, Rust, uma maneira de pensar sobre este problema é como um problema de engenharia de sistemas. Isto é, estabelecer a forma de particionar o sistema em dois subsistemas lógicos. Vamos chamar de Leste e Oeste, e alocar os requisitos para os subsistemas como se eles fossem sistemas físicos separados. Defina uma interface, complete com as definições de classes e serviços comuns a serem usados, isso permitirá que os dois subsistemas (aplicações) se cooperem para atingir a funcionalidade global do sistema.*

Rusty: *Mas eu não estaria criando um sistema arbitrariamente, que não esteja sendo dirigido pelos verdadeiros conceitos arquiteturais?*

Nós: *Verdade, no sentido técnico. Mas separar conceitos entre equipes de projeto alinhado à logística e especificar competências é mais importante.*

Rusty: *Mas eu não estaria criando interfaces artificiais e potencialmente, um sistema de chaminés?*

Nós: *Sim, faz sentido, mas nós recomendamos que você mantenha o mesmo código de interface em ambos os lados e desenvolvidos por uma única equipe. Caso contrário, as duas equipe realizarão trabalhos redundantes. Você irá, sem dúvida, criar novos requisitos para o sistema, incluindo interfaces que provavelmente não seriam necessárias. E, sim, é importante ter consciência do problema da chaminé e fazer tudo que puder para minimizar o acoplamento entre os sistemas e minimizar as questões políticas que surgirão como resultado.*

O Caso de Estudo

Enfim, após uma breve introdução à engenharia de sistemas, deixe nos aplicar o que aprendemos no sistema HOLIS, nosso Sistema de Automação de Iluminação Residencial. Neste momento, não gastaremos muito tempo tentando entender os requisitos do HOLIS. Nós faremos isso nos próximos capítulos. Nesse sentido, a engenharia de sistemas é prematura. Por outro lado, nós provavelmente entendemos o suficiente para tomar as primeiras decisões sobre o projeto do sistema, com base em nossa experiência, e em nosso entendimento do que sejam esses requisitos. Em muitos casos, nós não nos comprometemos ainda com qualquer coisa de hardware ou software; podemos retomar essas questões posteriormente. No processo iterativo que será descrito posteriormente, nós veremos a arquitetura de sistemas e requisitos de sistemas interativamente, de tal forma que este não é uma hora ruim para começar.

Necessidades Preliminares do Usuário

Deixe-nos assumir que já definirmos e entendemos, razoavelmente bem, algumas das necessidades do usuário do sistema HOLIS.

- O HOLIS precisará suportar chaves "soft" – chaves individualmente programáveis usadas para ativar as características de luminosidade nos diversos espaços da casa.
- O proprietário solicitou que houvesse algo para programar o HOLIS a partir de uma central remota, de tal forma que ele pudesse simplesmente chamar quando precisasse ao invés de ter que perder tempo em "programar" o HOLIS.
- Outros futuros compradores solicitaram que o HOLIS pudesse ser programado a partir de seus computadores pessoais e que fosse fornecida a habilidade de fazer, eles mesmos, todas as instalações, programações e manutenções.
- Outros, ainda, requisitaram que o sistema fornecesse um painel de controle simples – um tipo de interface simples onde eles pudessem mudar no HOLIS, a programação, configurar atividades de férias, entre outras, sem ter que usar o PC.
- O HOLIS precisa fornecer um sistema de contato para emergências de algum tipo.

Análise do Problema

Ao analisar o problema, a equipe decidiu, inicialmente, desenvolver três declarações do problema, um dos quais estabelece, aparentemente, o problema óbvio sob a perspectiva da empresa.

Elementos	Descrição
O problema	do baixo crescimento apresentado na principal área de atuação da empresa: iluminação profissional de teatros
afeta	a empresa, seus empregados e seus acionistas,
devido	ao desempenho inaceitável e substancial falta de oportunidades de crescimento em rendimento e lucratividade.
Os benefícios desse	novo produto e desse novo mercado em potencial para os produtos e serviços da empresa são: ▪ Revitalização da empresa e de seus empregados ▪ Elevação da lealdade e conservação dos distribuidores da empresa ▪ Alto rendimento e lucratividade ▪ Tendência de valorização das ações da empresa

Então a equipe também decidiu ver se podiam entender o "problema" a partir da perspectiva de um futuro cliente (usuário final) e potenciais distribuidores/ construtores (clientes da Lumenations):

Elementos	Descrição
O problema	da falta de opções de escolha de produtos, da funcionalidade limitada, e alto custo dos sistemas de iluminação de residências
afeta	os proprietários de sistemas residenciais de última geração
devido	ao desempenho inaceitável dos sistemas adquiridos ou, com maior freqüência, a decisão por não automatizar sua residência.
Os benefícios desse	sistema de automação para "correta" de iluminação são: ▪ Alta satisfação dos proprietários e orgulho de possuí-lo. ▪ Elevada flexibilidade e usabilidade da residência. ▪ Melhoria na segurança, conforto e conveniência.

Elementos	Descrição
O problema	a falta de opções para escolha de produtos, da funcionalidade limitada, e alto custo dos sistemas de iluminação de residências
afeta	os distribuidores e construtores de sistemas residenciais de última geração
devido	a poucas oportunidades de diferenciação no mercado e nenhuma nova oportunidade para aumentar a margem de lucro.
Os benefícios desse	sistema de automação para "correta" de iluminação são: ▪ Diferenciação. ▪ Alto rendimento e alto lucro. ▪ Aumento na participação de mercado.

HOLIS: O Sistema, Atores e Stakeholders

Deixe-nos voltar ao nosso projeto de engenharia de sistemas. Da perspectiva do sistema, a nossa primeira impressão do sistema HOLIS é simplesmente de um sistema dentro da residência do proprietário. A Figura 6–5 ilustra um simples diagrama mostrando o HOLIS no contexto da residência do proprietário.

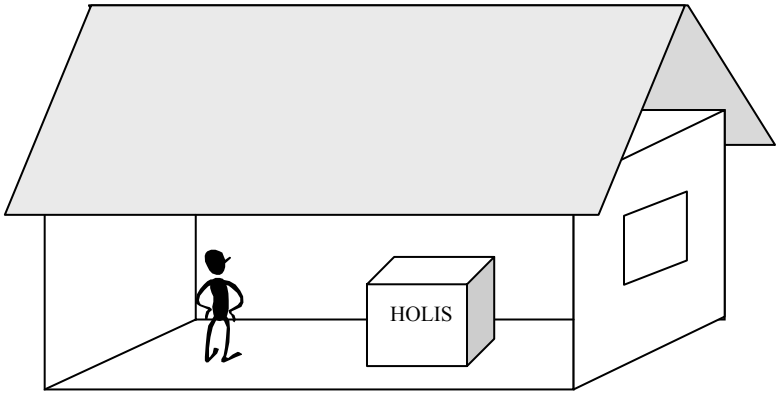


Figura 6–5 Contexto do sistema: HOLIS no seu ambiente

O passo 3 da análise do problema requer que nós *identifiquemos os stakeholders e usuários do sistema*. O passo 4 da análise do problema diz para *definir a fronteira da solução sistêmica*. Dado que as informações sobre as necessidades do usuário foram apresentadas, podemos agora aprimorar nosso entendimento do contexto do sistema HOLIS identificando os atores que irão interagir com o HOLIS. A Figura 6–6 ilustra 4 atores:

1. O proprietário que usa o HOLIS para controlar a luminosidade.
2. As várias lâmpadas que o HOLIS, por sua vez, controla.
3. Serviços da Lumenations, o fabricante que tem a habilidade para remotamente conectar-se ao HOLIS e realizar programações remotamente.
4. Receptor de Emergências, um ator indefinido que irá receber mensagens de emergência.

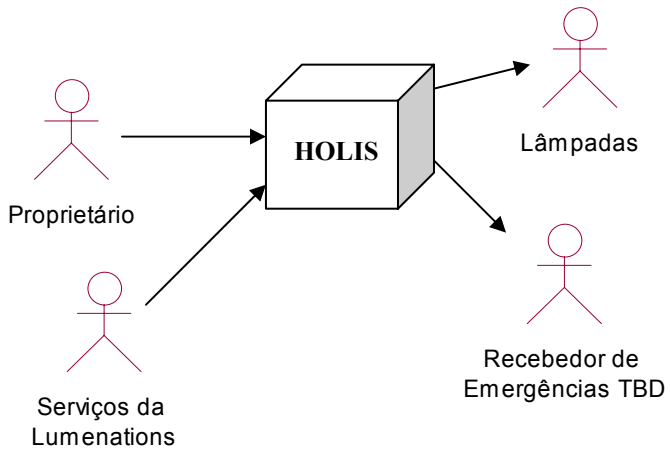


Figura 6–6 HOLIS com atores

Naturalmente, a equipe também descobriu vários stakeholders "não-atores", tanto internos quanto externos à empresa, preocupados com os requisitos do HOLIS, como mostra a Tabela 6–1.

Tabela 6–1 Stakeholders não-atores do HOLIS

Nome	Comentários
O Distribuidor Externo	Clientes diretos da Lumenations.
Construtores Eletricistas Contratados	Clientes dos Clientes da Lumenations: o contratado geral responsável pela construção da residência.
Equipe de Desenvolvimento Interno	Responsável pela instalação e suporte.
Gerente de Marketing/Produto	Equipe da Lumenations.
Gerente Geral da Lumenations	Será representado pela Cathy, gerente de produto.
	Financiamento e contabilidade dos resultados.

Engenharia de Sistemas do HOLIS

Agora que nós entendemos os atores externos do HOLIS, permita-nos fazer algumas considerações, em nível sistêmico, para ver como podemos particionar HOLIS em subsistemas. Este processo pode ser bem dirigido pelos seguintes tipos de pensamentos da engenharia de sistemas:

- Seria bom se nós pudéssemos ter software comuns tanto nos dispositivos de controle quanto no PC do proprietário; nós iremos pegar uma implementação baseada no PC para ambos os elementos do sistema.
- Nós ainda não estamos certos quanto à flexibilidade que estamos precisando nas chaves "soft" remotas, mas é claro que irão existir muitos deles, e que alguns irão estar distantes da unidade de controle principal, e que nós provavelmente precisaremos de alguma comunicação inteligente entre eles e a unidade de controle.

Com este pensamento minimalista, nós podemos surgir com uma nova perspectiva do sistema, aquela em que o HOLIS, o sistema, seja composto de três subsistemas: *Chaves de Controle*, o dispositivo de chaveamento remoto programável; *Unidade de Controle Central*, o sistema de controle de computador central; e o *Programador PC*, o sistema PC opcional que alguns proprietários requisitaram. Agora, o diagrama de bloco é apresentado na figura 6–7.

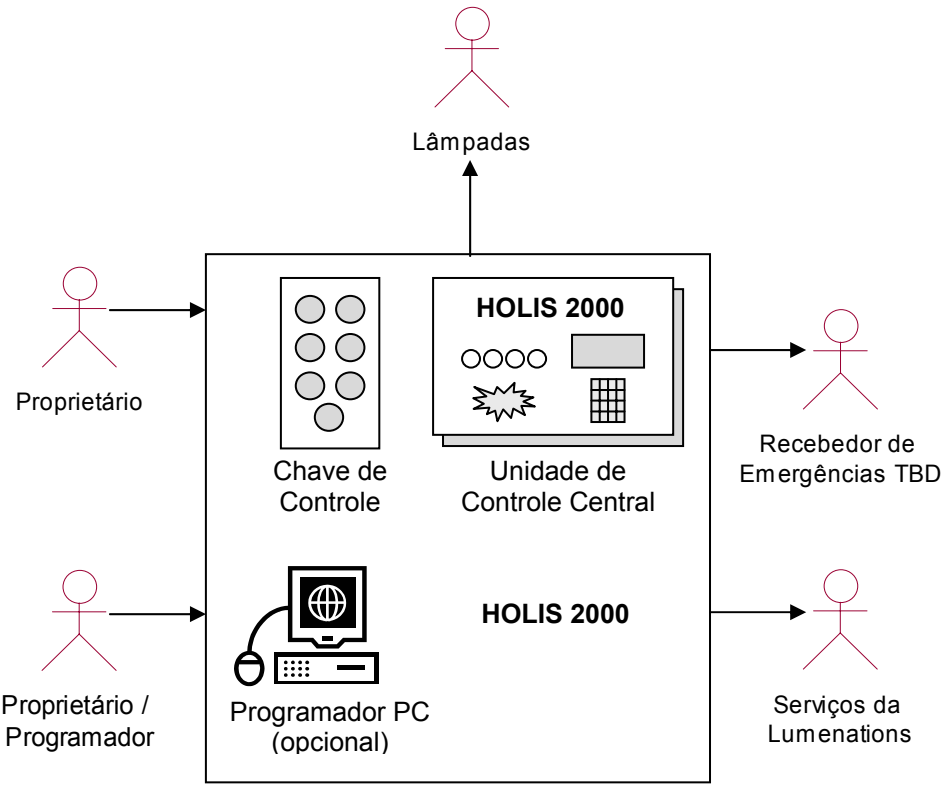


Figura 6–7 HOLIS com subsistemas e atores

Note que nós acabamos de identificar *cinco* atores – o proprietário novamente – mas que desta vez, usa o PC para programar o HOLIS ao invés de ligar e desligar lâmpadas. Este ator Proprietário/Programador possui, neste papel, diferentes necessidades para o sistema e assim é um ator distinto do sistema. Nós iremos ver, mais tarde, os diversos comportamentos que o ator espera do HOLIS.

Os Subsistemas do HOLIS

A partir da perspectiva da engenharia de sistema e de requisitos, o problema torna-se um pouco mais complexo. Além da necessidade de entender os requisitos do HOLIS, o sistema, nós iremos agora precisar, também, entender os requisitos únicos para cada um dos três subsistemas. Nós podemos usar nosso paradigma de ator novamente, no próximo nível de decomposição do sistema. Ao se fazer isso, surgem três novos diagramas de blocos: Figuras 6–8, 6–9, e 6–10.

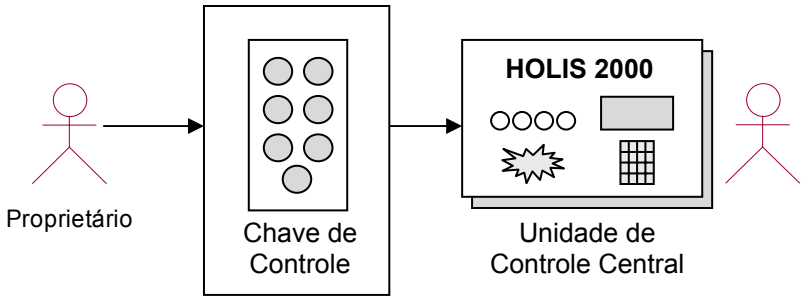


Figura 6–8 Subsistema Chave de Controle com seus atores

Na Figura 6–8, quando vemos sob a perspectiva da Chave de Controle, encontramos um outro ator: Unidade de Controle Central (UCC), um outro subsistema. Em outras palavras, da perspectiva do subsistema *Chave de Controle, o UCC é um ato*r, e precisaremos entender mais tarde quais tipos de requisitos e use cases a Chave de Controle terá que ter em função da UCC. Este conjunto de requisitos é derivado da nossa decomposição do sistema.

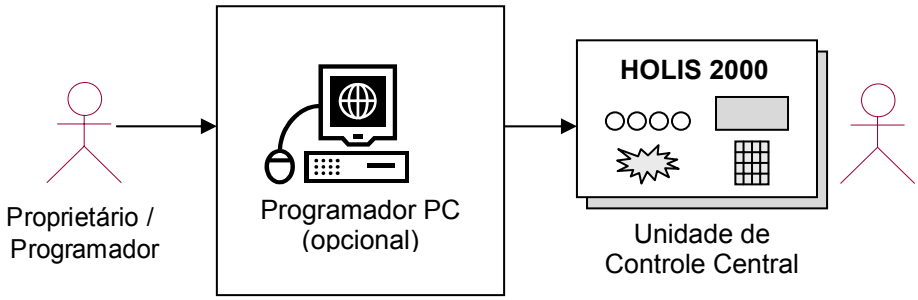


Figura 6–9 Subsistema Programador PC com seus atores

Na Figura 6–9, a perspectiva do sistema sob o ponto de vista do Programador PC, nós vemos que não aprendemos qualquer coisa nova, ao menos em termos de atores e subsistemas, eles foram todos identificados anteriormente. A Figura 6–10, no entanto, apresenta-se uma visão um pouco mais rica, e nós vemos que UCC possui mais atores do que qualquer outro. Isso parece fazer sentido, se nós pensarmos que a UCC é o cérebro do HOLIS, e faz sentido pensar que ele tem muitas coisas a fazer e muitos atores a atender.

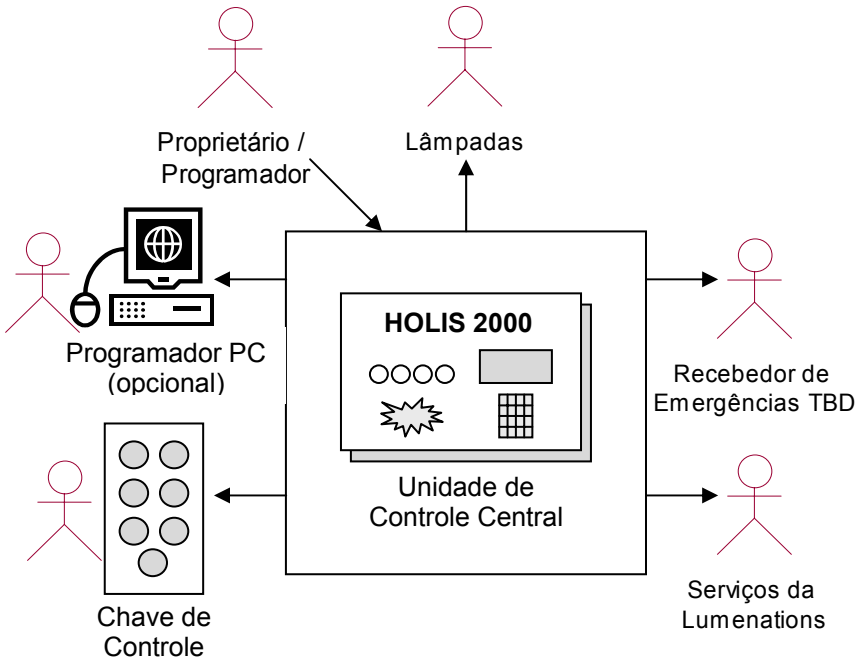


Figura 6–10 Subsistema Unidade Controle Central com seus atores

Tabela 6–2 Restrições do projeto HOLIS

#ID	Nome	Lógica
1	A versão 1.0 deverá ser liberada para manufatura em 5 de Janeiro de 2000.	A única oportunidade de lançamento do produto neste ano.
2	A equipe deve adotar a modelagem UML, métodos baseados em OO, e o Processo de Desenvolvimento Unificado.	Nós acreditamos que estas tecnologias fornecem aumento de produtividade e produzem sistemas robustos.
3	O software para a Unidade de Controle Central e o Programador PC devem ser escritos em C++. A linguagem Assembly será usada na Chave de Controle.	Devido à consistência e, também, à manutenibilidade, pois a equipe conhece estas linguagens.
4	Um protótipo do sistema deve ser apresentado numa exposição comercial de Automação Residencial em Dezembro.	Para obter pedidos de distribuidores do Q1 FY 2000.
5	O subsistema de microprocessamento da Unidade de Controle Central deve ser copiado da divisão profissional de projetos de sistemas de iluminação avança (ALSP).	É um projeto existente e uma peça existente em estoque.
6	Apenas o Programador PC deverá ser compatível com o Windows 98.	Faz parte do escopo de gerenciamento para a liberação da versão 1.0.
7	Contratar no máximo dois empregados, de tempo integral, somente após o término da fase de concepção, quando as habilidades necessárias ao projeto estarão determinadas.	Contratação máxima permitida para expansão da equipe.
8	O microprocessador KCH5444 deve ser usado na Chave de Controle.	Já em uso pela companhia.
9	Aquisições de componentes de software é possível, contanto que não exista nenhuma obrigação de pagamentos contínuos de royalty pela empresa.	Nenhum custo de longo prazo poderá causar impacto no custo de software.

Por agora, isto é o suficiente para a análise do problema e engenharia de sistemas do HOLIS. Nós iremos retornar a este caso de estudos nos próximos capítulos.

Sumário da Habilidade de Equipe 1

A Habilidade de Equipe 1, Analisando o Problema, introduziu um conjunto de habilidades que sua equipe pode aplicar para *entender o problema a ser resolvido antes de começar o desenvolvimento da aplicação*. Nós introduzimos de forma simples, os cinco passos da técnica de análise de problemas que podem ajudar a sua equipe a conseguir obter melhor entendimento do problema a ser resolvido.

1. Chegar ao acordo sobre a definição do problema.
2. Entender a causa raiz do problema – o problema por detrás do problema.
3. Identificar os stakeholders e usuários.
4. Definir a fronteira da solução sistêmica.
5. Identificar as restrições que serão impostas à solução.

A análise do problema desta forma sistemática, irá aperfeiçoar a habilidade de sua equipe em atacar futuros desafios – *fornecendo uma solução do problema a ser resolvido*.

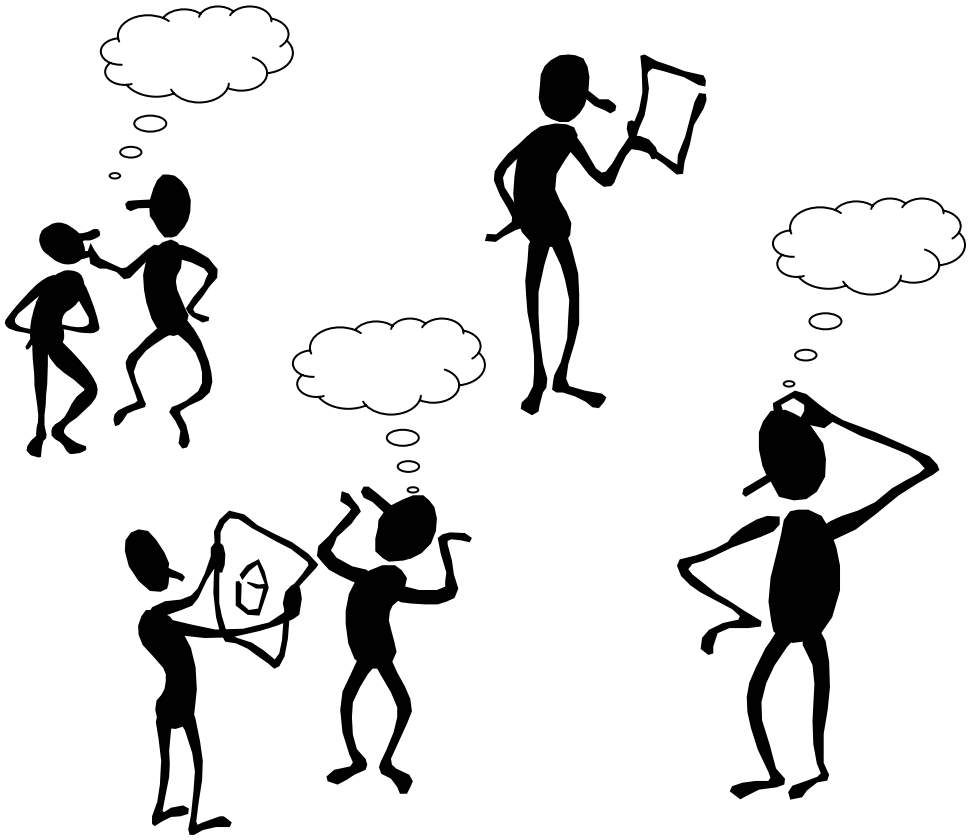
Nós também apontamos as várias técnicas que podem ser usadas na análise do problema. Em especial, vimos à modelagem de negócios, a qual funciona bem para sistemas de informação complexos que sustentam as principais infra-estruturas do negócio. A equipe pode usar a modelagem de negócios tanto para entender como o negócio funciona, quanto para definir onde o sistema, dentro do negócio, pode ser aplicado de forma mais produtiva. Reconhecemos também que o modelo de negócio que definimos possui construtores paralelos aos das aplicações de software, e usamos esta semelhança para reproduzir as fases de projeto de software. Nós iremos usar os use cases de negócio que descobrimos para ajudar a definir os requisitos da aplicação.

Para a classe de aplicações de software a qual denominamos de sistemas embutidos, usamos o processo de engenharia de sistemas como uma técnica de análise de problemas que nos ajuda a decompor sistemas complexos em subsistemas. Este processo nos ajuda a entender onde as aplicações de software deverão estar e qual o seu principal propósito. Ao fazer isso, aprendemos que complicamos o assunto de requisitos devido o surgimento de novos subsistemas, os quais devemos entender os requisitos a serem impostos.

Habilidade de Equipe 2

Entendendo as Necessidades dos Usuários

- **Capítulo 7**: Os Desafios da Elucidação de Requisitos
- **Capítulo 8**: As Características de um Produto ou Sistema
- **Capítulo 9**: Entrevistando
- **Capítulo 10**: O Workshop de Requisitos
- **Capítulo 11**: Brainstorming e Redução de Idéias
- **Capítulo 12**: Storyboarding
- **Capítulo 13**: Aplicando Use Cases
- **Capítulo 14**: Role Playing
- **Capítulo 15**: Prototipação



O fator mais comum entre os desafios de projetos foi "falta de informações de usuários" (Standish Group, 1994)

Um estudo publicado pela Standish Group citou a "Falta de informações de usuários" como o fator mais comum entre os desafios de projetos. Embora 13% dos entrevistados tivessem citado as causas raízes como o principal fator, 12% citaram as "especificações e requisitos incompletos" como fator principal. A partir disso, tornou-se aparente que a falta de entendimento das reais necessidades dos usuários (e provavelmente de outros stakeholders) representava mais de um quarto de todos os desafios de projeto, e era o problema mais sério que interferia para o sucesso do projeto.

A menos que, num belo dia, todos os usuários do mundo acordassem repentinamente, e começassem dispostos a entender, e a comunicar seus requisitos, a equipe de desenvolvimento terá que tomar sua iniciativa. Em outras palavras, nossa equipe precisará desenvolver a habilidade necessária para elucidar esses requisitos.

Problema

Na Habilidade de Equipe 1, nós desenvolvemos a habilidade que ajudará a entender o problema a ser resolvido. Na Habilidade de Equipe 2, nós descrevemos várias técnicas que a equipe de desenvolvimento pode usar para obter e entender as reais necessidades na perspectiva de usuários e stakeholders. Fazendo isso, começamos a ganhar compreensão dos potenciais requisitos do sistema que pretendemos desenvolver para satisfazer essas necessidades. Enquanto estivermos fazendo isso, nós iremos nos concentrar principalmente nas necessidades dos stakeholders, que vivem no topo da pirâmide de requisitos.

As técnicas que iremos ver variam desde a simples, inexpressivas, e extremamente simples, tal como a técnica de entrevista, até às mais modestas e caras tal como a técnica de prototipação. Embora nenhuma das técnicas seja perfeita para todas as situações, a exposição dessas várias técnicas irá fornecer um rico conjunto de habilidades a partir das quais a equipe poderá escolher. A cada projeto específico, a equipe poderá escolher entre as várias técnicas, aquela que achar mais apropriada e aplicá-la de acordo com a experiência e conhecimentos obtidos anteriormente na elucidação de requisitos em outros projetos. Desta forma, a equipe irá desenvolver um conjunto de habilidades que poderão contribuir ativamente para aperfeiçoamento dos resultados.

Capítulo 7

Os Desafios da Elucidação de Requisitos

Pontos chaves

- A elucidação de requisitos é influenciada por três síndromes endêmicas.
- A síndrome "Sim, mas" origina-se da natureza humana e da incapacidade dos usuários de perceberem o software da mesma forma que eles percebem os dispositivos de hardware.
- Procurar requisitos é como procurar por "Ruínas Desconhecidas"; quanto mais você encontra, mais você conhece.
- A síndrome do "Usuário e o Desenvolvedor" reflete a profunda diferença entre os dois, tornando difícil à comunicação.

Nos próximos capítulos, nós veremos uma variedade de técnicas para elucidar requisitos do sistema a partir de seus usuários e outros stakeholders[2]. Este processo é extremamente simples. Sente-se com os futuros usuários do sistema e com outros stakeholders e pergunte o que eles querem que o sistema faça.

Então, por que é tão difícil? Por que precisamos de tantas técnicas? Aliás, por que precisamos desta habilidade de equipe, afinal de contas? A fim de apreciar por completo este particular problema, deixe nos mostrar as três síndromes que parecem complicar imensamente esta matéria.

Obstáculos da Elucidação

A Síndrome do "Sim, Mas"

Um dos mais frustrantes, impregnantes e sinistros problemas do desenvolvimento de aplicações é o que nós conhecemos como a síndrome do "Sim, Mas", sendo observado nas reações de usuários para todas as peças de software que desenvolvemos. *Qualquer que seja a razão, sempre observamos duas reações imediatas, distintas e separadas quando o usuário vê a implementação do sistema pela primeira vez*:

- "Uau, que legal! Podemos realmente fazer isso! Fantástico; parabéns!", e assim por diante.
- "Sim, mas, hummmmm, como eu vejo isso? Para que serve esse ...? Não seria melhor se ...? O que acontece se ...?"

[2] Nós usamos o termo "usuário" genericamente neste contexto. As técnicas para elucidar requisitos do sistema se aplicam a todos os stakeholders, sejam usuários ou não-usuários.

As raízes da síndrome “Sim, Mas” parece repousar profundamente na natureza do software como um processo intelectual intangível. Para piorar ainda mais as coisas, nossa equipe de desenvolvimento normalmente *contribui* com o problema deixando de fornecer precocemente o código produzido aos usuários, para que eles possam interagir e avaliar os resultados.

A reação dos usuários faz parte da natureza humana e ocorre em várias outras circunstâncias do dia-a-dia. Os usuários que não viram o seu sistema ou qualquer outro similar, não entenderão o que você pensou quando descreveu o sistema, e agora que estão na frente do sistema – agora, pela primeira vez, depois de meses ou anos de espera – eles tem a oportunidade de interagir com o sistema. Agora, adivinhe o acontece? O sistema não é exatamente o que eles esperavam que fosse!

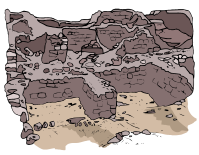
Por analogia, deixe-nos comparar este processo de software com o desenvolvimento de dispositivos mecânicos cujas tecnologias e processos de desenvolvimento antecedem o software por uns meros cem anos. Sistemas mecânicos possuem uma disciplina razoavelmente bem definida dos modelos de prova de conceitos, moldes, modelos, prototipação incremental, dispositivos de produção piloto, entre outros, todos eles possuindo aspectos tangíveis e a maioria é visível, perceptível e atua como dispositivos que está sendo desenvolvido.

Os usuários podem ver os dispositivos precocemente, tocá-los, pensar sobre eles, ou mesmo interagir com eles antes mesmo dos detalhes de implementação estarem finalizadas. É verdade que tecnologias específicas, tal como a litografia, onde um rápido protótipo é construído durante a noite a partir da injeção de um líquido viscoso a alta temperatura, foram desenvolvidas exclusivamente com o propósito de fornecer o feedback imediato e precoce da definição conceitual do produto. Apesar disso, no software, com sua enorme complexidade, esperamos fazer direito logo na primeira vez!

Embora possa ser frustrante, aceitar a síndrome “Sim, Mas” como uma realidade pode ajudar a discernir a realidade e auxiliar os membros da equipe a reduzirem esta síndrome em futuros projetos.

- A síndrome “Sim, Mas” faz parte da natureza humana e é parte integral do desenvolvimento de aplicações.
- Podemos reduzir drasticamente esta síndrome aplicando técnicas que busquem os “Mas” desde o início. Ao fazer isso, elucidamos as respostas do “Sim, Mas” no início, e então podemos começar a investir a maior parte dos esforços de nosso desenvolvimento no software que já tenha passado pelo teste do “Sim, Mas”.

A Síndrome das “Ruínas Desconhecidas”



Uma vez, um de nossos amigos foi ser guia de um ônibus de turismo na Four Corners area, uma área definida por fronteiras comuns dos estados do Colorado, Novo México, Utah e Arizona. A rota do ônibus de turismo incluía o majestoso pico da cadeia de montanhas La Plata, a extensão das antigas ruínas Anasazi do Mesa Verde e áreas próximas. As perguntas dos turistas são uma constante fonte de divertimento entre as equipes de guias turísticos e criam certos folclores no negócio de turismo. Numa estação de verão, nosso amigo nos contou qual era a

pergunta mais estúpida, realizada por um turista estúpido, da qual ele mais gostava: “Bem, hummmmm, quantos ruínas desconhecidas existem?”.

De certa forma, a busca por requisitos é como uma busca por ruínas desconhecidas: *Quanto mais os encontramos, mais os conhecemos*. Você nunca sentirá que encontrou todos, e talvez você nunca encontre. De fato, as equipes de desenvolvimento de software, de todas as partes do mundo, esforçam-se continuamente em determinar quando finalizar a elucidação de requisitos, isto é, determinar o momento em que eles terão encontrado todos os requisitos essenciais ou, ao menos, encontrado o suficiente.

A fim de auxiliar a equipe a atacar esse problema, nós fornecemos várias técnicas, tanto no capítulo Habilidade de Equipe 2 quanto nos posteriores. Naturalmente, como descrevemos no Capítulo 1, é muito importante gastar tempo em o analisar problema para identificar todos os stakeholders do sistema, porque muitos desses stakeholders não-usuários são normalmente detentores de requisitos desconhecidos. No entanto, da mesma forma que o problema de encontrar todas as ruínas desconhecidas, devemos adverti-lo de que estamos numa missão que pode nunca terminar. Mas entendemos também que em algum ponto, estaremos aptos a dizer com segurança, “Nós descobrimos o suficiente”.

A Síndrome “Usuário e o Desenvolvedor”

Técnicas de elucidação de requisitos não são novas. Desenvolvedores de aplicações têm as perseguido por mais de 40 anos para fazer um bom trabalho. Então, o que poderia explicar o fato de que entender as necessidades do usuário permanece um de nossos maiores problemas? Bem, considerando o fato de que alguns poucos desenvolvedores de aplicação tiveram treinamento em técnicas de elucidação, isso talvez não seja uma grande surpresa.

A terceira síndrome surge da *lacuna de comunicação* entre usuários e desenvolvedores. Chamamos esta síndrome de “Usuário e o Desenvolvedor”. Usuários e desenvolvedores são, normalmente, de mundos diferentes, falando linguagens diferentes e tendo diferentes experiências, motivações e objetivos.

De certo modo, nós devemos aprender a nos comunicar mais efetivamente com esses “usuários de outras tribos”. Num artigo sobre essa matéria, Laura Scharer (1981) descreve esta síndrome e fornece algumas orientações para ajudar a reduzir o problema. Combinando suas palavras com as nossas experiências, a Tabela 7–1 sumariza tanto as razões para este problema quanto sugere algumas soluções.

A esperança é que com um melhor entendimento, tanto da natureza desses problemas quanto das abordagens para mitigá-los, desenvolvedores possam estar melhores preparados para os futuros trabalhos de interesse.

Tabela 7–1 A síndrome "Usuário e o Desenvolvedor"

Problema	Solução
Usuários não sabem o que querem, ou sabem o que querem, mas não sabem dizer.	Reconhecer e apreciar o usuário como especialista do domínio; tentar técnicas alternativas de comunicação e elucidação.
Usuários pensam que sabem o que querem até que os desenvolvedores lhes digam o que eles disseram que queriam.	Fornecer técnicas de elucidação alternativas o quanto antes: storyboarding, role playing, protótipos descartáveis, entre outros.
Analistas pensam que entenderam o problema do usuário melhor do que os próprios usuários.	Coloque o analista no lugar do usuário. Faça com que o analista interprete o papel do usuário durante uma hora ou um dia.
Todos acreditam que todos estão politicamente motivados.	Sim, faz parte da natureza humana, então vamos continuar com o programa.

Técnicas de Elucidação de Requisitos

Alcançar melhor entendimento das necessidades do usuário leva-nos do domínio de bits e bytes, onde muitos desenvolvedores estão mais confortáveis, para o domínio das pessoas reais e problemas do mundo real. Da mesma forma que várias técnicas podem ser usadas para analisar e projetar soluções de software, várias técnicas podem ser usadas para entender necessidades de usuários e stakeholders. Algumas técnicas são seguidas por equipes de projeto em circunstâncias particulares.

Nos Capítulos 4, 5 e 6, nós iniciamos nosso passeio com a análise do problema, um conjunto de questões que podemos fazer sobre as restrições que serão impostas ao sistema, a técnica de modelagem de negócio que podemos usar para várias aplicações, e a técnica de engenharia de sistemas que podemos aplicar para sistemas complexos. Nos capítulos seguintes, descreveremos técnicas que provaram ser efetiva em atacar as três síndromes que acabamos de discutir. Entre as técnicas que iremos discutir estão:

- Entrevistas e questionários
- Workshops de requisitos
- Brainstorming e redução de idéias
- Storyboards
- Use cases
- Role playing
- Prototipação.

A escolha de uma técnica específica varia, dependendo do tipo de aplicação, a habilidade e sofisticação da equipe de desenvolvimento, a habilidade e sofisticação do cliente, a escala do problema, urgência da aplicação, a tecnologia usada, e da exclusividade da aplicação.

Capítulo 8

As Características (Features) de um Produto ou Sistema

Pontos chaves

- A equipe de desenvolvimento deve interpretar um papel mais ativo na elucidação dos requisitos do sistema.
- As características (features) do produto ou sistema são expressões de alto nível do comportamento desejado do sistema.
- Características do sistema devem estar limitadas a 25-99, com menos que 50 preferivelmente.
- Atributos fornecem informações adicionais sobre as características.

Após apresentar alguns problemas nos capítulos anteriores, ficou claro que a equipe de desenvolvimento raramente, se não nunca, tem em mãos uma especificação perfeita, ou talvez razoável, que possa ser usada como base para o desenvolvimento do sistema. No Capítulo 7, aprendemos sobre as razões disso. Uma conclusão que chegamos foi que, se nós não nos prepararmos para dar as melhores definições, nós não iremos colher os resultados. Em outras palavras, a fim de obter sucesso, a equipe de desenvolvimento terá que interpretar mais ativamente seu papel na elucidação de requisitos. Como descobrimos, embora possamos delegar a maioria das responsabilidades para um líder sênior, analista, ou gerente de produto, no final, toda a equipe estará envolvida com um ou mais pontos do processo.

Stakeholders e Necessidades do Usuário

Parece óbvio, mas a equipe de desenvolvimento somente irá construir sistemas melhores quando eles entenderem as reais necessidades (needs) dos stakeholders. Esse entendimento dará à equipe a informação que necessita para tomar melhores decisões na definição e implementação de sistemas. Esse conjunto de informações, o qual chamamos de *necessidades (needs) de stakeholders*, ou simplesmente de *necessidades do usuário*, fornece a peça central do quebra-cabeça.

Normalmente, as necessidades do usuário são vagas e ambíguas. Por exemplo, seu stakeholder pode dizer: "Eu preciso de algo fácil para saber o estado do meu estoque" ou "Eu gostaria de ver um grande aumento de produtividade nas entradas dos pedidos de venda". Mesmo assim, essas declarações definem o contexto mais importante de todas as outras atividades que virão. Por serem tão importantes, gastaremos algum tempo e energia tentando entendê-los. Nós definimos uma necessidade (needs) do stakeholder como:

uma reflexão de negócio, pessoal ou de problema operacional (ou de oportunidade) que deve ser atendida a fim de justificar a consideração, compra ou uso de um novo sistema.

Características (Features)

Uma coisa interessante que ocorre, quando entrevistamos stakeholders para conhecer suas necessidades ou os requisitos para um novo sistema, é que normalmente, esses stakeholders não falam nada disso, ao menos em termos das definições que fornecemos anteriormente. Esses stakeholders não dizem a você nenhuma de suas reais necessidades – "Se eu não aumentar a produtividade deste departamento, eu não ganharei meu bônus este ano" ou "Eu quero reduzir a velocidade deste veículo o mais rápido que puder sem derrapar" – nem os reais requisitos do sistema – "Eu preciso reduzir o tempo de processar as transações de entrada dos pedidos de venda em 50%" ou "O veículo deve ter um sistema de controle computadorizado para cada roda". Ao invés disso, eles descrevem o que acham ser uma abstração de algo como "Eu necessito de uma nova tela de entrada de pedidos baseado em GUI" e "Eu quero um veículo com ABS".

Denominamos essas expressões de alto nível sobre o comportamento desejado do sistema de *características (features)*. Essas características não são, normalmente, bem definidos e podem até, serem conflitantes entre si. "Eu quero aumentar a média de processamento de pedidos" e "Eu quero fornecer uma interface amigável para ajudar os novos empregados aprenderem a usar o sistema" – mas eles são, no entanto, uma representação das reais necessidades.

O que está acontecendo nesta discussão? O stakeholder já traduziu a necessidade real (produtividade e segurança) no comportamento do sistema o qual ele acha que irá atender às suas necessidades (veja Figura 8–1). Ao fazer isso, o *o que* ("Eu necessito") foi substituído pelo *como* ("O que eu penso que o sistema deva fazer para satisfazer esta necessidade"). Isso não é uma coisa ruim, nas vezes quando o usuário tem real especialidade do domínio e real percepção do valor da característica. Também, por causa da facilidade de se discutir tais características em linguagem natural, documentá-las e comunicá-las aos outros, adicionam tremenda riqueza ao esquema de requisitos.

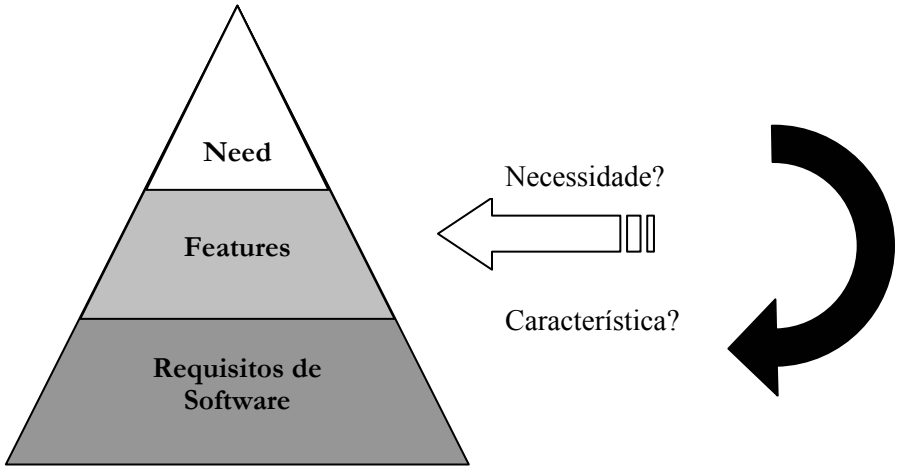


Figura 8–1 As necessidades e características estão intimamente relacionadas

Características são formas convenientes de descrever as funcionalidades sem ter que se atolar em detalhes.

No entanto, existe uma interrupção no processo desta discussão, qual seja: Se a equipe abandonar a discussão sem um entendimento das reais necessidades por detrás das características, então haverá um real risco. Se as características não resolverem as reais necessidades por alguma razão, então o sistema poderá falhar em atender os objetivos dos usuários, mesmo que a implementação tenha sido derivada das características que foram exigidas.

Em qualquer caso, nós encontramos esta abstração de alto-nível – essas características – são formas úteis e convenientes de descrever funcionalidades de um novo sistema sem se atolar em detalhes. De fato, nós dirigimos a maior parte de nossas atividades de requisitos a partir desta idéia de característica.

No começo, definimos uma característica como:

um serviço que o sistema fornece para atender um ou mais necessidades dos stakeholders.

Com esta definição, características de usuários não podem estar tão longe das necessidades e temos uma maneira cômoda de iniciar a definição do sistema.

Nosso foco está em entender as necessidades dos usuários a fim de elucidar e organizar as *necessidades* e *características* do sistema proposto. Algumas vezes, obtermos todas as necessidades e nenhuma característica. Algumas vezes obtemos todas as características e nenhuma necessidade. Algumas vezes nós não somos capazes de perceber a separação. Mas, desde que tomemos cuidado em distingui-los em nossas mentes, poderemos, a todo o momento, aprender informações valiosas sobre o que o sistema deve fazer.

A características são facilmente descritas em linguagem natural e consiste de uma frase curta; alguns exemplos são mostrados na Tabela 8–1. Raramente, ou nunca, características são elaboradas em maiores detalhes. Características são também construtores muito úteis para dar início do gerenciamento do escopo do projeto, para estabelecer negociações e processos de acordo. A declaração de características não exige uma grande quantidade de investimento, e são fáceis de descrevê-las e relacioná-las.

Tabela 8–1 Exemplos de características

Domínio da Aplicação	Exemplo de uma Característica (Feature)
Sistema de controle de elevador	Controle manual de portas durante incêndios.
Sistema de controle de estoque	Fornecer o estado de todos os itens de estoque.
Sistema de rastreamento de defeitos	Fornecer informações de tendência e qualidade do produto
Sistema de pagamento	Relatório de deduções por data e por categoria.
HOLIS (Sistema de automação de iluminação de residências)	Configuração para longos períodos de ausências.
Sistema de controle de armas	Obrigatoriedade de no mínimo duas confirmações de ataque.
Aplicação de uma copiadora	Compatibilidade com o Windows 2000.

Gerenciando a Complexidade Escolhendo o Nível de Abstração

Características do sistema devem estar limitadas a 25-99, com menos que 50 preferivelmente.

O número de características que consideramos para nós mesmos irá efetivamente determinar o nível de abstração da definição. Para gerenciar a complexidade de sistemas que estamos vislumbrando, seja de um novo sistema ou incremento de um sistema existente, *recomendamos que existam entre 25 a 99 características, abstraídas em nível suficientemente alto, sendo melhor pouco menos de 50*.

Desta forma, uma quantidade relativamente pequena e gerenciável de informações fornecem a base compreensiva e completa para a definição do produto, a comunicação com os stakeholders, o gerenciamento de escopo, e o gerenciamento de projeto. Com 25 a 99 características convenientemente categorizadas e organizadas, podemos estar aptos a descrever e comunicar a estrutura: do sistema, do ônibus espacial ("reentrada e reuso") ou de uma ferramenta de software ("tendência automática de defeitos").

Em Habilidade de Equipe 5, tais características serão transformadas em requisitos específicos, suficientemente detalhadas para permitir a implementação. Nós iremos chamar aqueles de *requisitos de software* para diferenciá-los de outras características de alto-nível. Nós iremos lidar com estas necessidades para incrementar a especificação posteriormente. Por agora, no entanto, nós iremos manter nosso pensamento em nível de características.

Atributos das Características do Produto

A fim de ajudar a melhor gerenciar esta informação, introduzimos a noção de *atributos* de características, ou elementos de dados que fornecem informações adicionais sobre o item. Atributos são usados para associar a característica ou dados de requisitos a outros tipos de informações de projeto. Nós podemos usar atributos para rastrear (nome ou identificador único, estado, dados históricos, alocação, rastreado para, entre outros), para priorizar (campo de prioridade), e para gerenciar (estado) as características propostas para a implementação. Por exemplo, o atributo *prioridade* pode ser usado para capturar os resultados da votação cumulativa numa sessão de brainstorming; o atributo *número da versão* pode ser usado para registrar as liberações de software específicas com as características específicas que pretendemos implementar.

Por anexar vários atributos às características, você pode gerenciar melhor a complexidade da informação. Embora não existam limites para os tipos de atributos que você pode ter, a experiência tem demonstrado que alguns atributos comuns de características se aplicam a muitas circunstâncias de projeto (Tabela 8–2). No restante deste volume, usaremos esses atributos para ajudar a gerenciar a complexidade dos dados de características e requisitos e para gerenciar relacionamentos, tais como as dependências entre os vários tipos de requisitos do sistema.

Assim, deixe-nos prosseguir com algumas habilidades de equipe que irá nos ajudar a obter as informações que nós precisamos. Nós começaremos com a entrevista (Capítulo 9).

Tabela 8–2 Atributos de características

Atributo	Descrição
Estado	Rastreia o progresso durante a definição do baseline do projeto e desenvolvimento subseqüentes. *Exemplo*: Proposto, Aprovado e Incorporado.
Prioridade / Benefício	Nenhuma característica é criada da mesma forma. Classificação por prioridade relativa ou beneficio para o usuário final abre um diálogo entre stakeholders e membros da equipe de desenvolvimento. Usado no gerenciamento de escopo e determinação de prioridade. *Exemplo*: Crítico, Importante e Útil.
Esforço	Estimar o número da equipe – ou pessoas-mês, linhas de código ou pontos de função, ou apenas o nível geral do esforço ajuda configurar expectativas de o que pode e não pode ser executado num dado quadro de tempo. *Exemplo*: Baixo, Médio e Alto.
Risco	Uma medida de probabilidade que a característica irá causar eventos indesejáveis, tais como exceder custo, atrasar o cronograma, ou mesmo cancelamento. *Exemplo*: Alto Médio e Alto.
Estabilidade	Uma medida de probabilidade de que a característica irá mudar ou de que o entendimento da equipe sobre as características que irão mudar. Usado para ajudar a estabelecer prioridades de desenvolvimento e determinar aqueles itens para os quais elucidação adicional é a próxima ação apropriada.
Meta de liberação	Registro das versões pretendidas de produtos, onde as características aparecerão pela primeira vez. Quando combinado com o campo Estado, sua equipe pode propor, registrar e discutir várias características sem liberá-las para o desenvolvimento.
Associado a	Em muitos projetos, características estarão associadas a "equipes de características" responsáveis em promover a elucidação, escrever os requisitos de software, e talvez realizar a implementação.
Razão	Usado para rastrear a fonte das solicitações de características. Por exemplo, a referência pode ser a página e o número da linha de uma especificação do produto ou um marcador de minuto num vídeo de uma importante entrevista do cliente.

Capítulo 9
Entrevista

Pontos chaves

- A entrevista é uma técnica simples e direta.
- Questões livres de contexto podem nos ajudar a conduzir com tranqüilidade as entrevistas.
- Então, a entrevista pode ser apropriada para encontrar requisitos não descobertos, por explorar soluções.
- Convergências sobre algumas necessidades comuns irão iniciar um "repositório de requisitos" para ser utilizado durante o projeto.
  Um questionário não substitui uma entrevista.

Uma das técnicas mais importantes e mais simples de obtenção de requisitos é a entrevista de usuários, uma técnica simples e direta e que pode ser usada em, virtualmente, qualquer situação. Este capítulo descreve o processo de entrevista e fornece um modelo genérico para conduzir a entrevista de usuários e stakeholders. No entanto, o processo de entrevista não é fácil, pois nos força a uma convivência próxima e pessoal e a enfrentar a síndrome "Usuário e o Desenvolvedor".

Além disso, uma das metas chaves da entrevista é assegurar que a propensão e a predisposição do entrevistador não interfira com a livre troca de informações. Este é um problema sutil e pernicioso. Professores de sociologia (opa, uma outra classe de profissional que nós evitamos!) nos ensinam que é *impossível* haver relacionamentos entre pessoas sem o filtro do mundo, que é o resultado de nosso próprio ambiente e experiências acumuladas.

Além disso, como fornecedores de solução, raramente nos encontramos numa situação em que não temos idéia alguma sobre os tipos soluções possíveis que possam atacar o problema. De fato, em muitos casos, operamos dentro de um domínio ou contexto recorrente onde certos elementos da solução são óbvios, ou ao menos nos parecem óbvios. ("Nós já resolvemos esse tipo de problemas antes, e esperamos que nossa experiência se aplique totalmente neste caso. Afinal, estamos apenas construindo casas, e martelos e pregos atenderão bem a esse propósito"). Naturalmente, isso não é ruim, porque o nosso contexto é parte do que temos de valor. Nosso ponto é que não devemos deixar que o contexto interfira no entendimento do real problema a ser resolvido.

O Contexto da Entrevista

As Questões livres de contexto

Então, como evitar que nossas questões prejudiquem a resposta do usuário? Nós o fazemos formulando questões sobre a natureza do problema do usuário, livre de

Uma questão livre de contexto nos ajuda a entender os reais problemas sem influenciar o usuário.

qualquer contexto da potencial solução. Para tratar deste problema, Gause e Weinberg (1989) introduziram o conceito de "questões livres de contexto". São exemplos de tais questões:

- Quem é o usuário?
- Quem é o cliente?
- As necessidades são diferentes?
- Onde mais podemos encontrar soluções para este problema?

Estas questões nos forçam a ouvir antes de tentar inventar ou descrever uma possível solução. Enquanto ouvimos, entendemos melhor o problema do cliente, bem como quaisquer problemas por detrás dos problemas. Tais problemas afetam a motivação e o comportamento do nosso cliente e devem ser atacados antes que possamos apresentar uma solução.

Questões livres de contexto possuem um paralelo com as questões ensinadas aos vendedores como parte de uma técnica chamada "soluções de venda". Nas soluções de venda, os vendedores usam uma série de questões focalizadas sobre a obtenção inicial do entendimento do problema do cliente e quais soluções, se existe algum, o cliente já anteviu. A intenção dessas questões é permitir que o vendedor entenda totalmente o real problema do cliente, tal que soluções efetivas possam ser sugeridas e determinadas de acordo com os seus méritos específicos. Este processo ilustra o valor das técnicas de venda como um dos ingredientes da solução completa para o real problema do cliente.

A Importância do Contexto da Solução

Depois que as questões livres de contexto forem formuladas, sugira soluções que possam ser exploradas.

Em nossa busca por descobrir requisitos, pode também ser apropriado fazer questões sobre o domínio onde as soluções são exploradas depois que as questões livres de contexto tiverem sido realizadas e respondidas. Afinal de contas, a maioria de nós, normalmente, não nos satisfazemos somente em entender o problema, mas em fornecer soluções apropriadas para o problema a ser resolvido. Esta adição do contexto da solução pode dar ao usuário, novas percepções e talvez até uma visão diferente do problema. E, naturalmente, nossos usuários dependem de nós para ter o contexto; caso contrário, eles teriam que nos ensinar todas as coisas que eles conhecem sobre o assunto.

Como um auxílio à construção desta habilidade dentro da equipe de desenvolvimento, nós combinaremos essas técnicas dentro da nossa "entrevista livre de contexto genérica", uma entrevista estruturada que pode ser usada para elucidar solicitações de usuários e stakeholders em muitos contextos da aplicação de software. O modelo para esta entrevista é fornecido na Figura 9–1. A entrevista consiste tanto de seções livres de contexto quanto de seções não-livres de contexto. Também fornece questões designadas para nos assegurar que todos os aspectos dos requisitos, incluindo alguns daqueles requisitos "te enganei" de segurança, suportabilidade, entre outros, tenham sido perfeitamente explorados.

Parte I: Estabelecendo o Perfil do Cliente ou Usuário

Nome:
Empresa:
Indústria:
Ocupação:
(As informações acima podem, normalmente, ser preenchidas antecipadamente).
Quais são as suas responsabilidades?
Quais são os produtos do seu trabalho?
Para quem você gera esses produtos?
Como é medido o seu sucesso?
Quais são os problemas que interferem para o sucesso de seu trabalho?
O que, se existe, facilita ou dificulta o seu trabalho?

Parte II: Avaliando o Problema

Para quais problemas faltam boas soluções "(tipo de aplicação)"?
Quais são? (Dica: Continue perguntando, "Mais algum?").
Para cada problema, pergunte:

- Por que esse problema existe?
- Agora, como resolvê-lo?
- Como você poderia resolvê-lo?

Parte III: Entendendo o Ambiente do Usuário

Quem são os usuários?
Qual é a sua formação?
Qual é a sua experiência em computação?
Os usuários são experientes nesse tipo de aplicação?
Quais plataformas são usadas?
Quais são os planos para a futura plataforma?
Outras aplicações usadas são relevantes para essa aplicação? Se sim, fale um pouco sobre elas.
Quais são as suas expectativas para a usabilidade do produto?
Quais são as suas expectativas para o tempo de treinamento?
Que tipo de auxílio ao usuário você precisa (p.ex., cópia impressa ou documentos on-line).

Parte IV: Recapturando para Entender

Você me disse que:
(Liste os problemas que o cliente descreveu com suas próprias palavras).

-
-
-

Eles representam adequadamente os problemas que você está tendo com a solução existente?
Quais outros problemas, caso exista, você está enfrentando?

Parte V: Suposições do Analista sobre o Problema do Cliente

(Valide ou invalide suposições).
(Se não foi discutido) Quais problemas, se houver, estão associados: (Liste quaisquer necessidades (needs) ou problemas adicionais que você acha que está preocupando o cliente ou o usuário).

-
-
-

Para cada problema sugerido, pergunte:

- Esse é um problema real?
- Quais são as razões deste problema?
- Como você gostaria de resolvê-lo?
- Qual é o peso da solução desse problema, comparado aos outros que você mencionou?

Parte VI: Avaliando a sua Solução (se aplicável)

(Resuma as principais capacidades da solução que você propôs).
O que aconteceria se você conseguisse:

-
-

Como você classificaria cada uma dessas capacidades, por ordem de sua importância?

Parte VII: Avaliando a Oportunidade

Quem na sua organização precisa dessa aplicação?
Quantos usuários desse tipo usariam a aplicação?
O que você considera que seja uma solução bem sucedida?

Parte VIII: Avaliando Necessidades (needs) de Segurança, Desempenho e Suportabilidade

Quais são suas expectativas sobre a segurança?
Quais são suas expectativas sobre o desempenho?
Você irá dar suporte ao produto ou serão outras pessoas que farão isso?
Você tem necessidades (needs) especiais de suporte?
O que você pensa sobre a manutenção e serviços de rede?
Quais são os requisitos de segurança?
Quais são os requisitos de instalação e configuração?
Existem requisitos especiais de licenciamento?
Como o software será distribuído?
Existem requisitos de etiquetagem ou de empacotamento?

Parte IX: Outros Requisitos

Existe algum requisito legal, de regulação, ambiental ou de padronização que deva ser atendido?
Você acha que existem outros requisitos que devemos conhecer?

Parte X: Fechamento

Existe alguma outra questão que eu deveria ter feito?
Se depois, eu tiver alguma dúvida, posso ligar para você? Você concorda em participar de uma revisão de requisitos?

Parte XI: Resumo do Analista

Depois da entrevista, e enquanto as informações estiverem frescas em sua mente, resuma as três necessidades (needs) ou problemas de maior prioridade identificados pelo usuário/cliente.

1.
2.
3.

Figura 9–1 A entrevista livre de contexto genérica

O Momento da Verdade: A Entrevista

Com um pouco de preparação e com a entrevista estruturada no bolso, qualquer membro da equipe pode fazer um trabalho adequado de entrevistar um usuário ou cliente. (Mas, seria melhor escolher membros da equipe que sejam mais extrovertidos). Aqui estão algumas dicas para o sucesso da entrevista.

- Preparar uma entrevista livre de contexto apropriadamente, e anotar numa agenda para servir de referência durante a entrevista. Revise as questões um momento antes da entrevista.
- Antes da entrevista, pesquise o histórico dos stakeholders e da companhia a ser entrevistada. Não sonde as pessoas sendo entrevistadas com questões que você poderia ter respondido antecipadamente. Por outro lado, não será prejudicial fazer uma *breve* verificação dessas respostas com o entrevistado.
- Anote respostas numa agenda durante a entrevista. (Não tente coletar os dados eletronicamente nesse momento!).
- Consulte o modelo durante a entrevista para se assegurar que as questões corretas estão sendo perguntadas.

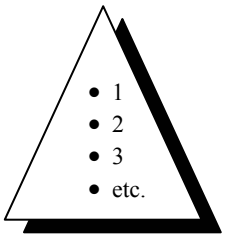
O entrevistador deve garantir que o roteiro não crie constrangimento ao entrevistado. Depois que a aproximação tenha sido bem sucedida, a entrevista provavelmente seguirá o seu próprio curso. Os clientes podem cair num diálogo de fluxo-de-consciência, descrevendo com detalhes sangrentos, o horror da situação atual. *É esse, exatamente, o comportamento que você está procurando*. Se acontecer com você, não interrompa prematuramente com outras questões; ao invés disso, escreva tudo o mais rápido possível, permita que o usuário descarregue esse fluxo de pensamento em particular! Siga formulando perguntas sobre as informações que acabaram de ser fornecida. Então, depois que essa linha de pensamento tenha alcançado o seu final lógico, retorne a outras questões da sua lista.

Depois de algumas entrevistas desse tipo, o desenvolvedor/analista terá obtido algum conhecimento do domínio do problema e terá chegado ao entendimento tanto do problema a ser resolvido quanto das percepções do usuário sobre as características de uma solução efetiva. Além disso, o desenvolvedor pode sumarizar as principais necessidades (*needs*) do usuário ou características (*features*) do produto que foram definidos na entrevista. Essas "necessidades (*needs*) do usuário" vivem no topo de nossa pirâmide de requisitos e servirão de guia para nos orientar em todas as tarefas seguintes.

Compilando os Dados de Necessidade (Needs)

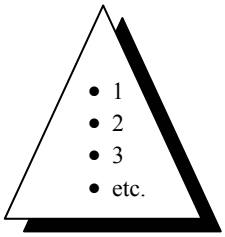
A sua análise do problema terá identificado os principais stakeholders e usuários que você precisará para entrevistar e para obter o entendimento das necessidades (needs) dos stakeholders. Normalmente, não precisamos fazer muitas entrevistas para obter um bom e sólido entendimento sobre o assunto.

O Resumo do Analista: 10 + 10 + 10 ≠ 30



A última seção do formulário de entrevista, Resumo do Analista, é usada para registrar as "três necessidades (needs) ou problemas mais importantes" descobertas na entrevista. Em muitos casos, após algumas entrevistas, tais necessidades (needs) de alta prioridade começarão a se repetir. Isso significa que você começou a encontrar convergência sobre algumas necessidades (needs) comuns. Isso é esperado, especialmente entre aqueles usuários e stakeholders que compartilham das mesmas perspectivas. Então, em dez entrevistas, normalmente, são criados apenas 10–15 necessidades (needs) diferentes. Este é o início do seu "repositório de requisitos", um conjunto de recursos que você irá construir e usar com grandes vantagens no decorrer do projeto. Esses dados simples e inexpressivos, por si só, ajudarão você e a sua equipe a construir uma base mais sólida para dar início ao seu projeto.

O Estudo de Caso



Needs do usuário do HOLIS

A equipe do HOLIS decidiu que a equipe de marketing (Eric e Cathy) desenvolveria as questões para a entrevista, mas procura alguém da equipe para experimentar o processo e ter a oportunidade de encontrar clientes face-a-fase e assim "ver" o problema e uma provável solução sob a perspectiva do cliente. Assim, a equipe dividiu a lista de clientes e distribuidores e cada membro da equipe teve que entrevistar duas pessoas. A equipe usou o Resumo do Analista para sumarizar as necessidades (needs) que foram fornecidas e duplicidades foram extraídas. Depois de quinze entrevistas, a equipe identificou 20 – algumas necessidades (needs) que preencherão o topo de nossa pirâmide de requisitos.

Da Perspectiva dos Proprietários:

- Controle de iluminação residencial modificável e flexível.
- "à prova do futuro" ("Como as tecnologias mudam, eu gostaria que houvesse compatibilidade com novas tecnologias que possam emergir.").
- Atrativo, discreto e ergonômico.
- Independência total e chaves programáveis ou reconfiguráveis para cada cômodo da residência.
- Mais segurança e despreocupação.
- Operação intuitiva ("Eu quero ensinar minha mãe tecnofóbica").
- O sistema com custo razoável, com chaves de baixo custo.
- Correção fácil e barata.
- Configuração de chaves flexível (de um a sete botões por chave).
- Fora do alcance da visão e observação.
- 100% confiável.
- Configuração de segurança de férias.
- Habilidade de criar cenas, como configuração de iluminação para festas.
- Nenhum incremento elétrico ou de perigo de incêndio na casa.
- Habilidade de após uma falta de energia elétrica, restaurar a iluminação.
- Conseguir programar usando o meu PC.
- Escurecer onde eu quiser.
- Conseguir programar sem o meu PC.
- Alguém mais irá programá-lo para mim.
- Se o sistema falhar, eu ainda quero estar apto a ligar algumas luzes.
- Interfaces para o meu sistema de segurança residencial.
- Interface com outros aparelhos (HVAC, áudio/vídeo, entre outros).

Da Perspectiva dos Distribuidores:

- Um produto que ofereça competitividade.
- Alguma forte diferenciação do produto.
- Facilidade de treinar o pessoal de vendas.
- Poder ser demonstrado em minha loja.
- Alta margem bruta.

Uma Nota sobre Questionários

Não existe substituto para a entrevista.
- Execute-a primeiro!
- Execute-a, para cada nova classe de problemas!
- Execute-a, para cada novo projeto!

Nós nos perguntamos com freqüência se a equipe pode substituir este processo de entrevista por um questionário. Em alguns casos, isso expressa apenas um desejo de eficiência ("Eu poderia fazer 100 questionários no tempo em que se leva para fazer uma única entrevista"). Em outros casos, isso pode expressar algum desejo suspeito ("Eu realmente tenho que falar com essas pessoas? Eu não poderia apenas enviar uma carta?").

Independentemente da motivação, a resposta é *não*. Não existe nenhum substituto para o contato pessoal, construção do entendimento e interação em formato livre da técnica de entrevista. Nós lhe asseguramos que após um ou duas entrevistas, nossa visão do mundo terá mudado. Mais importante que isso, a visão da solução terá mudado junto com ele! Primeiro, executa a entrevista. Execute-a, para cada nova classe de problema! Execute-a, para cada novo projeto.

No entanto, quando usada apropriadamente, a técnica do questionário pode também exercer um papel legítimo na obtenção das necessidades (needs) do usuário. Embora a técnica do questionário seja freqüentemente usada e pareça científico por causa da oportunidade de se fazer análise estatística de resultados quantitativos, a técnica não substitui a entrevista. Quando aplicada na obtenção de requisitos, a técnica do questionário apresenta alguns problemas fundamentais:

- Questões relevantes podem não ser levada adiante.
- As suposições por detrás das questões influenciam as respostas.

  Exemplo: Esta classe atende a suas expectativas? Suposição: Que a pessoa possui expectativas.

- É difícil explorar novos domínios (O que você realmente deveria ter perguntado é ...), e não existe interação para explorar domínios que precisam ser explorados.
- Respostas não claras do usuário são difíceis de perseguir.

De fato, alguns concluíram que a técnica do questionário suprime quase todas as coisas boas da obtenção de requisitos, e assim, nós geralmente não a recomendamos para este propósito.

Questionários podem ser usados para validar suposições e obter dados estatísticos sobre preferências.

No entanto, a técnica do questionário pode ser aplicada com bons resultados após a entrevista inicial e atividades de análise. Por exemplo, se a aplicação tiver vários usuários em potencial ou existente, e se a meta é obter informações estatísticas das preferências de usuários ou clientes entre um conjunto limitado de escolhas, o questionário pode ser efetivo. Para finalizar, a técnica do questionário, como toda técnica de elucidação, satisfaz a um conjunto de desafios de requisitos que uma organização pode enfrentar.

Capítulo 10

Workshops de Requisitos

Pontos chaves

- Talvez o workshop de requisitos seja a técnica mais poderosa para se elucidar requisitos.
- Reúne todos os principais stakeholders por um breve porém intenso período.
- O uso de um facilitador externo e experiente em gerenciamento de requisitos pode ajudar a assegurar o sucesso do workshop.
- O brainstorming é a parte mais importante do workshop.

Acelerando o Processo de Decisão

Um dos esquemas de priorização que usamos é enganosamente simples. Pergunte ao stakeholder qual característica (feature) ele escolheria caso tivesse que escolher apenas uma, a qual seria implementada na próxima liberação (release). Como na mente de alguém que vai para a forca, esta questão faz com que a mente maravilhosamente concentre-se no que é realmente importante.

Quando houver apenas uma única oportunidade de elucidar requisitos – uma que possamos aplicar em todas as circunstâncias, sem se importar com o contexto do projeto, sem se importar com o período de tempo – escolha o workshop de requisitos. O workshop de requisitos pode ser muito bem, ser a técnica mais poderosa deste volume e uma das poucas que, quando dominado, pode realmente ajudar a mudar os resultados do projeto, mesmo que seja a única técnica aplicada.

O workshop de requisitos é projetado para alcançar consenso sobre requisitos da aplicação e chegar rapidamente ao acordo sobre o curso das ações, tudo num período de tempo muito curto. Nesta técnica, os principais stakeholders do projeto são reunidos por um breve e intensivo período, normalmente não mais que 1 ou 2 dias. O workshop é facilitado por um membro da equipe ou, melhor ainda, por um facilitador externo experiente concentrado na criação ou na revisão das características (features) de alto-nível que a nova aplicação deve ter.

Uma execução apropriada do workshop de requisitos trás muitos benefícios:

- Ele ajuda na construção de uma equipe efetiva, comprometida com um único propósito comum: o sucesso deste projeto.
- Todos os stakeholders têm a sua vez de falar, ninguém fica de fora.
- Ele produz um acordo entre stakeholders e a equipe de desenvolvimento sobre o que a aplicação deve fazer.
- Ele pode expor e resolver assuntos políticos que estão interferindo para o sucesso do projeto.

- O resultado, que é uma definição preliminar do sistema em nível de características (features), é imediatamente disponibilizado.

Muitas empresas têm alcançado sucesso com a técnica de workshop. Juntos, nós participamos em mais de 100 desses workshops, e raramente, se houve algum, houve insucesso em atingir os objetivos desejados. O workshop fornece oportunidade única para stakeholders, de várias partes da empresa, trabalharem juntos com o objetivo comum de alcançar o sucesso do projeto.

Neste capítulo, você irá aprender a planejar e executar com sucesso, um workshop de requisitos. No final do capítulo, nós aplicaremos esta técnica no nosso caso de estudos HOLIS.

Preparando o Workshop

A *preparação* apropriada do workshop é crítica para o seu sucesso.

Vendendo o Conceito

A preparação apropriada do workshop é crítica para o seu sucesso.

Primeiro, pode ser necessário vender o conceito dentro da empresa, comunicando os benefícios da abordagem workshop aos futuros membros da equipe. Este processo normalmente não é difícil, mas é comum encontrar resistências: "Não, mais uma reunião!"; "Provavelmente não conseguiremos reunir todas essas pessoas críticas num único dia"; "Você nunca será atendido pelo [nome do seu stakeholder favorito]". Não se desencoraje; se você acreditar, eles virão.

Assegurando a Preparação dos Stakeholders Corretos

Segundo, a preparação também envolve a identificação dos stakeholders que podem contribuir para o processo e cujas necessidades (needs) devem ser atendidas para assegurar o sucesso dos resultados. Esses stakeholders já terão sido identificados se a equipe executou o passo da análise do problema, mas agora é tempo para uma última revisão para garantir que todos os stakeholders críticos foram identificados.

Logísticas

Terceiro, é necessária uma abordagem consciente para a logística e pagará dividendos na medida em que um workshop seja miseravelmente organizado; certamente, os resultados desejados não serão atingidos. A logística envolve todas as coisas, desde estruturar apropriadamente os convites, viagens, acomodações, até a iluminação da sala de reuniões do workshop. Uma crença literal nas leis de Murphy – "Se pode dar errado, dará errado" – deve ser nosso guia aqui. Se você abordar a logística com um alto grau de profissionalismo, é óbvio que atenderá aos propósitos num importante evento, e eles irão atuar de acordo. Você terá, também, mais sucesso no workshop.

"Material de Aquecimento"

Quarto, envie materiais do workshop antecipadamente para preparar os participantes e também elevar a produtividade nas sessões de workshop. Esses materiais preparam o estado de espírito dos participantes. Chamamos isso de "atingir um ideal estado de espírito". Uma das mensagens que nós precisamos passar é que *esta não é mais uma reunião. Esta pode ser a única chance de dizer isso de forma correta*.

Material de aquecimento estimula tanto o pensamento de quem faz parte do contexto, quanto de fora do contexto (out-of-box).

Recomendamos que você forneça dois tipos separados de materiais de aquecimento:

1. *Informações específicas do projeto*, o qual pode incluir o esboço do documento de requisitos, lista de características (features) sugeridas, cópias de entrevistas com os futuros usuários, relatório do analista sobre tendências na indústria, cartas de clientes, relatório de erros do sistema existente, novas diretivas de gerenciamento, novos dados de marketing, entre outros. Embora seja importante não sufocar com dados os nossos futuros participantes, é importante assegurar que eles recebam os dados corretos.
2. *Preparação do pensamento out-of-box*. Parte do "atingir um ideal estado de espírito" está em encorajar o pensamento "out-of-the-box". "Esqueça por um minuto o que você conhece e o que não pode ser feito devido à política". "Esqueça o que você tentou colocar em atividade na última vez e falhou". "Esqueça que nós não solidificamos nosso processo de desenvolvimento". "Apenas traga suas sugestões sobre características (features) deste novo projeto e esteja preparado para pensar 'out of the box'". Como líder do workshop, você pode ajudar neste processo fornecendo pensamentos provocativos e estimulando o acordo sobre o processo de criatividade, regras para o brainstorming, gerenciamento de requisitos, gerenciamento de escopo, entre outros. Nesta atmosfera, soluções criativas serão as mais prováveis.

Dica: Não envie dados com muita antecedência. Você não irá querer que os participantes leiam e esqueçam o que leram, e você não irá querer que um longo ciclo de planejamento reduza o sentido de urgência. Envie os dados entre 2 dias e 1 semana de antecedência. De qualquer forma, é muito provável que os participantes só lerão o plano no último minuto. Tudo bem; isso ajudará a preparar o estado de espírito para a sessão.

Para ajudar você com o pensamento "out-of-box" e auxiliar a configurar o contexto das atividades do workshop, nós fornecemos um modelo de um memorando na Figura 10–1. Entre parênteses, nós iremos também "ler entre linhas" partes do texto para entender alguns desafios que você pode enfrentar em seu projeto, e sobre como o workshop pretende atacá-lo.

Memorando:
Para: **Stakeholder do projeto _____**
Assunto: **Realização do Workshop de Requisitos**
De: [Nome do Líder de Projeto]

Eu sou o gerente de projeto [produto]. O projeto foi [ou irá ser] iniciado em _____ e será finalizado na data de _____.

(Nós sabemos; nós entendemos, e pretendemos finalizar nesse período).

Como em muitos projetos, temos dificuldades de chegar a um consenso sobre as novas características desta aplicação e em definir uma versão inicial que atenda as necessidades de nossos diversos grupos de stakeholders.

(É muito difícil chegar a um acordo sobre qualquer coisa com este grupo, então nós estamos tentando algo um pouco diferente, e aqui está o que isso é ...)

A fim de facilitar este processo, nós estamos organizando um workshop de requisitos em _____.

O objetivo deste workshop é de fechar as novas características para a próxima versão do produto. A fim de fazer isso, é importante ouvir a contribuição de todos os stakeholders. O workshop será conduzido pelo _____, que tem grande experiência como facilitador no gerenciamento de requisitos.

(Como stakeholder, podemos também ter preconceitos, então chamamos alguém de fora da equipe para nos assegurarmos que o workshop será gerenciado livre de qualquer preconceito).

Os resultados deste workshop serão disponibilizados imediatamente e serão distribuídos para as equipes de desenvolvimento e marketing no dia seguinte. Convidamos vossa senhoria a participar deste workshop como representante das necessidades de vossa [equipe, departamento, cliente]. Se não for possível a vossa participação, nós solicitamos que envie um membro de sua equipe que esteja autorizado a tomar decisões como representante de suas necessidades.

(Nós iniciaremos o desenvolvimento nos próximos dias; se você quiser ouvir contribuições para este projeto, esteja aqui, ou envie alguém que possa falar por você. Em outras palavras, fale agora ou descanse em paz).

Em anexo, a este memorando, está uma breve descrição antecipada das características do produto, bem como algum material de leitura sobre o processo de workshop e brainstorming. O workshop começará exatamente às 8:30 horas e terminará até as 17:30 horas.

(Este projeto e este workshop serão executados de forma profissional; para demonstrar isso, nós fornecemos alguns materiais antecipadamente para melhor prepará-lo. Nós precisamos que você esteja aqui, para contribuir, e nos ajudar a conduzir este projeto de maneira apropriada)
.
Certo de poder contar com a vossa participação, agradeço antecipadamente.

Cordialmente,

[Nome do Líder de Projeto]

Figura 10–1 Exemplo de memorando para rápido início de um workshop de requisitos

Papel do Facilitador

Para garantir o sucesso, nós recomendamos que o workshop seja executado por alguém de fora e que tenha experiência em enfrentar os desafios do processo de gerenciamento de requisitos. No entanto, se isso não for uma alternativa prática em seu ambiente, o workshop pode ser facilitado por um membro da equipe, desde que esta pessoa:

- Tenha recebido algum treinamento neste processo.
- Tenha solidamente demonstrado habilidade de construir consenso ou equipe.
- Seja elegante e bem respeitado tanto pelos membros internos quanto externos.
- Seja forte o suficiente para conduzir a reunião a qual pode ser um encontro desafiador.

Se possível, tenha um facilitador que não seja da equipe para executar o workshop.

No entanto, se o workshop for facilitado por um membro da equipe, essa pessoa *não deve* contribuir com idéias e assuntos da reunião. Caso contrário, o workshop estará em grave perigo de perder a objetividade que é necessária para obter os fatos reais, e pode não estimular um verdadeiro ambiente onde o consenso possa emergir.

Em alguns casos, o facilitador tem o papel central para o sucesso do workshop. Afinal de contas, você está com todos os principais stakeholders reunidos, talvez pela primeira vez e última vez no projeto, e você não pode se permitir errar o alvo. Algumas das responsabilidades do facilitador são:

- Estabelecer um tom profissional e objetivo para a reunião.
- Iniciar e finalizar a reunião dentro do tempo.
- Estabelecer e impingir as "regras" da reunião.
- Introduzir as metas e a agenda da reunião.
- Gerenciar a reunião e manter a equipe "na trilha".
- Facilitar o processo de decisão e o consenso, mas evitar participar do conteúdo.
- Gerenciar qualquer instrumento e assuntos de logística para assegurar que o foco permaneça na agenda.
- Assegurar que todos os stakeholders participem e sejam ouvidos.
- Controlar comportamentos desordeiros ou improdutivos.

Preparando a Agenda

A agenda do workshop terá como base as necessidades do particular projeto e no conteúdo que deve ser desenvolvido no workshop. Ninguém preenche toda a agenda. No entanto, workshops de requisitos estruturados podem seguir um formato padrão claro. A Tabela 10–1 fornece uma agenda típica.

Tabela 10–1 Exemplo de agenda para o workshop de requisitos

Hora	Item de Agenda	Descrição
8:00 – 8:30	Introdução	Agenda, recursos e regras.
8:30 – 10:00	Contexto	Estado do projeto, necessidade do mercado, resultados da entrevista do usuário, etc.
10:00 – 12:00	Brainstorning	Características do brainstorning da aplicação.
12:00 – 13:00	Almoço	Trabalhar durante o almoço para evitar perder o momento.
14:00 – 15:00	Definição da característica	Escrever 2 ou 3 definições de sentença para as características (features).
15:00 – 16:00	Redução e priorização de idéias	Priorizar as características (features).
16:00 – 17:00	Fechamento	Sumário e itens de ação, atacar os itens de "estacionamento".

Executando o Workshop

Problemas e Truques do Ofício

Você pode ver que o facilitador interpreta um papel crucial. Para tornar o assunto mais interessante, esses workshops são caracterizados, normalmente, por uma atmosfera altamente carregada. Em outras palavras, existem razões para que seja difícil chegar ao consenso nos projetos; quase todas aparecerão no workshop.

De fato, o cenário pode até estar carregado e conflituoso politicamente. Esta é a outra razão para ter um facilitador; deixar o facilitador aquecer e gerenciar a reunião de forma que não exacerbe qualquer problema – seja do passado, presente, ou do futuro – entre os stakeholders.

Muitos facilitadores carregam uma "mala de truques" que o auxiliam a gerenciar esta atmosfera altamente carregada. Na RELA, desenvolvemos um conjunto muito útil de "truques de workshop". Embora eles pareçam estranhamente bonitos, e até infantil a primeira vista, você pode acreditar, eles provaram o seu valor em várias situações. Quanto mais difícil o workshop, mais valioso será! Eles também tendem a estimular o pensamento "out-of-box". O mais interessante é que eles são divertidos e contribuem para o tom positivo da sessão. A Figura 10–2 fornece um exemplo desse conjunto de truque de workshop. Sinta-se livre para adaptá-los e usá-los, junto com as "instruções" de uso.

A Tabela 10–2 descreve algum dos problemas que podem ocorrer numa preparação do workshop e também fornece sugestões sobre como você pode usar os truques de workshop para atacar os problemas. O facilitador deve também introduzir essas regras no início da reunião e, idealmente, obter aprovação consensual para aplicar essas tolas regras por um dia.

Regra: Inicialmente, cada participante recebe um cupom grátis quando chegar atrasado. Depois disso, o participante contribui com $1 de multa.

Regra: Inicialmente, cada participante recebe um cupom para "dar um toque" sobre uma pessoa ou departamento. Depois disso, o participante contribui com $1 de multa.

Objetivo: Fazer com que a pessoa fique ciente dos assuntos políticos do projeto de forma divertida.

Regra: Inicialmente, cada participante inicia com dois cupons. Os participantes dão o cupom para qualquer participante que forneça uma grande idéia. O objetivo é gastar seus cupons.

Objetivo: Incentivar e gratificar pensamentos criativos.

Regra: O participante gasta este cupom em algum momento. O facilitador dá a palavra ao participante e marca seu tempo. Todos ouvem. Não há interrupção.

Objetivo: Viabilizar a participação organizada. Assegurar que todos dêem sua opinião.

Figura 10–2 Truques de workshop

Tabela 10–2 Problemas e soluções na configuração do workshop de requisitos

Problema	Solução
Gerenciamento do tempo • É difícil reiniciar as atividades após os intervalos. • Os principais stakeholders podem se atrasar.	O facilitador mantém controle do tempo de serviços de cozinha em todos os intervalos. Participantes que chegarem atrasados devem contribuir com um cupom "Atraso após o intervalo" ou pagar $1 para a caixinha de "contribuições para caridade".
Discurso exagerado e dominador, visando impressionar e receber ovação do público.	O facilitador reforça a regra "5 minutos para falar" para regular o discurso. Ele também cria uma lista de assuntos pendentes para, posteriormente, poder voltar às idéias que mereçam ser discutida, mas que não são relevantes ao item do momento de acordo com a agenda.
Carência de contribuições dos stakeholders.	O facilitador encoraja os participantes a usarem seus cupons de "5 minutos para falar" e "Que grande idéia!". Ele deixa claro que ninguém deve deixar o workshop sem ter usado os cupons ou terem recebido um cupom "Que grande idéia!" de outros. (Sugestão: Dê uma pequena premiação para quem usar o cupom "5 minutos para falar", dando-lhe um cupom "Que grande idéia!").
Comentários negativos, comportamento de baixo escalão e clima de guerra.	Use o cupom "1 Ofensa gratuita e deliberada" até que os participantes não tenham mais nenhum; depois disso, têm que dar contribuições para a caixinha de "contribuições para caridade". (o grupo decide quanto).
Energia debilitada após os intervalos.	Almoço leve, cafezinhos rápidos, rearranjo da sala, rearranjo dos locais dos participantes, alterar a luminosidade e temperatura. Faça o que você puder fazer para manter as coisas funcionando.

Brainstorming e Redução de Idéias

A parte mais importante do workshop é o processo de brainstorming. Esta técnica é idealmente adaptada para o ambiente de workshop, e estimula uma atmosfera criativa e positiva e estimula contribuições de todos os stakeholders. Nós iremos tratar do brainstorming no próximo capítulo.

Produção e Continuidade

Depois do workshop, o facilitador distribui os minutos da reunião para registrar algumas outras informações. Assim, o trabalho do facilitador está terminado, e a responsabilidade do sucesso volta novamente para a equipe de desenvolvimento.

Depois disso, o líder de projeto tem a responsabilidade de dar continuidade a quaisquer itens de ação em aberto que foram registradas na reunião e organizá-los para distribuir aos participantes. Normalmente, as informações da reunião apresentam-se como uma lista de idéias ou características sugeridas para produto que podem modificar imediatamente as futuras ações da equipe de desenvolvimento. Em alguns casos, workshops adicionais com outros stakeholders serão programados, ou esforços adicionais de elucidação serão necessários para ganhar melhor entendimento das idéias estimuladas no workshop. A criação dessas idéias é o centro de todo o processo do workshop. No próximo capítulo veremos mais de perto esta parte do processo.

Capítulo 11

Brainstorming e a Redução de Idéias

Pontos chaves

- O brainstorming trata tanto da geração de idéias quanto da redução de idéias.
- As idéias mais criativas e inovadoras normalmente resultam da combinação múltipla, aparentemente de idéias desassociadas.
- Várias técnicas de votação podem ser usadas para priorizar as idéias criadas.
- Embora seja preferível o brainstorming presencial, o brainstorming baseado em web pode ser uma alternativa viável em algumas situações.

Se você está configurando um workshop, como vimos no Capítulo 10, ou se você procura por novas idéias ou soluções criativas para problemas, o brainstorming é uma técnica muito útil. É simples, fácil e divertido.

Na configuração do workshop, você provavelmente já terá uma boa idéia sobre as características do novo produto. Afinal de contas, poucos projetos começam com passado totalmente limpo. Dessa forma, além de revisar as características sugeridas do produto, o workshop fornece a oportunidade de solicitar novas informações, mudar e combinar essas novas características com aquelas que já estavam sob consideração. Este processo irá nos ajudar na nossa meta de "encontrar as ruínas desconhecidas" e através disso, assegurar a integridade das informações e que todas as necessidades dos stakeholders foram cobertas. Tipicamente, uma parte do workshop é dedicada ao brainstorming de novas idéias e características da aplicação. O Brainstorming é uma coleção de técnicas que são úteis quando os stakeholders estão lado-a-lado.

Esta técnica de elucidação tem como benefícios principais:

- Encorajar a participação de todas as partes presentes.
- Permitir que os participantes "engatem" uma idéia com a outra.
- Registro de todas as coisas discutidas pelo facilitador ou secretário (assim, nada é perdido).
- Alta quantidade de informações.
- Normalmente, gera um grande conjunto de possíveis soluções para qualquer que seja o problema proposto.
- Encoraja o pensamento "out-of-the-box", isto é, sem as limitações das restrições normais.

O brainstorming tem duas fases: geração de idéias e redução de idéias. O objetivo principal durante a geração é delinear tantas idéias quanto sejam possíveis; o objetivo é ampliar as idéias, não necessariamente aprofundá-las. O objetivo

principal da redução de idéias é aparar, organizar, priorizar, expandir, agrupar, refinar, e assim por diante.

Brainstorming Presencial

Embora o brainstorming possa ser abordado de diferentes maneiras, o processo simples que descrevemos provou ser efetivo em vários cenários. Inicialmente, todos os stakeholders importantes são reunidos numa sala e recursos são distribuídos. Os recursos distribuídos a cada participante podem ser tão simples quanto um bloco de folhas de anotações e um marcador preto grosso para escrever as anotações. As folhas devem ter, ao menos, 3" × 5" (7 cm × 12 cm) e não maior que 5" × 7" (12 cm × 17 cm). Cada participante deve ter ao menos 25 folhas para cada sessão de brainstorming. Post-Its ou cartões de índices funcionam bem. Se cartões de índice são usados, são necessários alfinetes e quadro de isopor ou cortiça.

Então, as regras do brainstorming são apresentadas (veja a Figura 11-1), e o objetivo da sessão é claramente e concisamente estabelecidos.

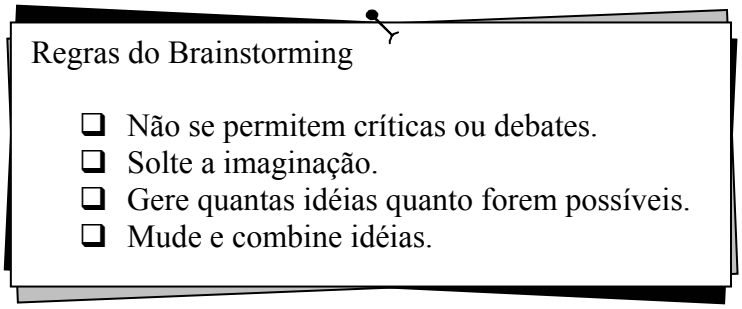

Regras do Brainstorming
Não se permitem críticas ou debates.
Solte a imaginação.
Gere quantas idéias quanto forem possíveis.
Mude e combine idéias.

Figura 11–1 Regras do brainstorming

O facilitador também expõe os objetivos do processo. Embora os objetivos que estabelecem o processo possam parecer muito simples, não é. A maneira como é estabelecido tem grande efeito sobre as conseqüências da sessão. Como exemplo, as seguintes questões dão algumas das formas de estabelecer os objetivos:

- Quais características você gostaria que o produto tivesse?
- Quais serviços você gostaria que o produto fornecesse?
- Que informações você gostaria que o produto mantivesse?

(Note que os objetivos também ajudam você a decidir quando a sessão está terminada. Quando os objetivos forem alcançados e não houver mais nada a acrescentar, fim!)

Depois que os objetivos do processo tiverem sido estabelecidos, o facilitador convida os participantes para que expressem suas idéias em voz alta e escreva-as, uma em cada folha. Idéias são ditas em voz alta para permitir que outros possam "engatar" suas idéias, isto é, pensar em relacionar idéias e seguir a regra 4, mudar e combinar idéias. Neste processo, no entanto, a primeira regra – sem críticas ou debates – deve estar à frente na mente das pessoas. Se esta regra não for impingida, o processo será silenciado, muitas pessoas brilhantes, porém, sensíveis às críticas, não se sentirão confortáveis em colocar suas idéias, uma trágica perda.

Dica: Em nossa experiência, as idéias mais criativas e inovadoras – aquelas que verdadeiramente revolucionam o conceito do produto – não resultaram da idéia de uma única pessoa, mas, ao invés disso, do resultado de múltiplas combinações, e aparentemente não-relacionadas, de idéias de vários stakeholders. Qualquer processo que estimule este resultado é um processo realmente poderoso.

Quando surge uma idéia, ela é registrada no material fornecido pela pessoa que teve a idéia. Isso é importante:

- Para assegurar que sejam registradas com as palavras da própria pessoa.
- Para assegurar que não sejam perdidas.
- Para permitir que sejam aproveitadas posteriormente.
- Para prevenir atrasos no processo criativo que pode ser provocado por um único secretário tentando capturar todas as idéias num flipchart ou quadro branco em frente à sala.

Assim que as idéias são geradas, o facilitador coleta e as afixa sobre uma parede da sala de reuniões. Novamente, nenhuma crítica às idéias pode ser tolerada. Não é apropriado dizer, "Que idéia estúpida!", ou mesmo "Já temos essa idéia afixada na parede!". O único propósito é gerar idéias. Mesmo um pacífico comentário negativo pode ter o deletério (danoso) efeito de suprimir futuras participações da "vítima". Por outro lado, comentários tais como "Que grande idéia!", são apropriados e são normalmente premiados com o cupom "Que grande idéia!", o qual pode encorajar futuras participações de todos os stakeholders. A geração de idéias deve prosseguir até que todos os participantes sintam que tenham chegado naturalmente ao seu final.

É comum ocorrer calmaria durante a geração de idéias. Isso não é motivo para parar o processo. A calmaria tende a cessar assim que uma nova idéia é gerada. Longas calmarias podem fazer com que o facilitador repasse os objetivos ou faça perguntas associadas. Muitas sessões de geração de idéias duram por volta de uma hora, mas algumas duram de 2 a 3 horas. Sob nenhuma condição deve o facilitador finalizar uma sessão que esteja caminhando bem com comentários como: "Eu sei que estamos todos fazendo grandes progressos, mas precisamos prosseguir para a próxima fase". Para os participantes, isso soa como "Nossas idéias não são tão importantes quanto o cronograma". A quantidade de idéias geradas depende de quão fértil foi a discussão, mas é comum gerar de 100 a 200 idéias.

O processo tende a terminar naturalmente; em algum ponto, os stakeholders ficarão sem idéias novas. Isso é caracterizado por um longo e grande intervalo entre submissões de idéias. Nesse ponto, o facilitador finaliza a sessão e pode ser um bom momento para fazer uma pausa.

Redução de Idéias

Quando a fase de geração de idéias termina, é a vez de iniciar a redução de idéias. Vários passos estão envolvidos.

Expurgando

O primeiro passo é o expurgo daquelas idéias que não são tão valiosas para merecer maior investimento pelo grupo. O facilitador inicia visitando brevemente cada idéia e solicitando a cooperação do grupo para indicar as idéias que acham serem válidas. Não existe razão para que qualquer participante fique na defensiva ou reivindique autoria de qualquer idéia; qualquer participante pode apoiar ou refutar qualquer idéia.

Dica: A presença de idéias que podem ser facilmente expurgadas é uma indicação do processo de qualidade. A ausência de uma quantidade razoável de idéias sem nexo e malucas indica que os participantes não pensaram no "out of the box" suficientemente.

O facilitador apenas pergunta aos participantes se cada uma das idéias merece futura consideração e então simplesmente remove uma idéia inválida, mas se houver qualquer desacordo entre os participantes, a idéia fica na lista. Se participantes acharem duas anotações com a mesma idéia, agrupá-las na parede. (É preferível fazer isso do que remover uma; seu autor pode se sentir insultado).

Agrupando Idéias

Pode ser útil durante este processo, iniciar agrupando idéias similares. Isso é feita de forma mais efetiva quando participantes voluntários se dirigem à parede e realizam o agrupamento. Idéias relacionadas são agrupadas em regiões da parede. Dê nomes aos grupos de idéias relacionadas. Por exemplo, o grupo pode ser rotulado como:

- Novas características
- Assuntos de desempenho
- Evolução das características existentes
- Interface de usuário e assuntos de usabilidade

Ou, podem concentrar especificamente nas capacidades do sistema e na maneira como eles suportam os vários tipos de usuários. Por exemplo, na previsão de um novo serviço de transporte e entrega, as características poderiam ser agrupadas em:

- Roteamento e rastreio de pacotes
- Serviços aos clientes
- Marketing e vendas
- Serviços baseados na web
- Faturamento
- Gerenciamento de transporte

A geração de idéias pode ser reiniciada agora para qualquer um desses grupos se os participantes acharem que o processo de agrupamento tenha estimulado o desenvolvimento de novas idéias ou que algumas áreas de funcionalidades-chave tenham sido deixadas de lado.

Definição de Características

Neste ponto, é importante gastar algum tempo para escrever uma breve descrição sobre o que as idéias significam para a pessoa que a submeteu. Isso dá ao contribuidor a oportunidade de apresentar características adicionais e ajudar a assegurar que os participantes tenham o mesmo entendimento dessas características. Desta maneira, todos entenderão o significado da idéia, evitando se assim, uma grande falha no processo de priorização.

Neste processo, o facilitador visita cada idéia que não foi expurgada e solicita ao contribuidor que forneça uma descrição em uma única sentença.

Contexto da Aplicação	Característica Obtida no Brainstorming	Definição da Característica
Automação de iluminação residencial	"Configuração da iluminação automática"	O proprietário pode criar, previamente, cronogramas para certos eventos com base nas horas do dia.
Sistema de entrada de pedidos de venda	"Rápido"	Rápido o suficiente para que o tempo não interfira nas operações normais.
Sistema de rastreamento de defeitos	"Notificação automática"	Todas as partes registradas serão notificadas via e-mail quando algo for mudado.

Uma característica de robô de solda como "soldagem automática", pode ter sido suficientemente descrita, não necessitando de nenhum trabalho adicional. No entanto, é importante não cair nessa armadilha; a descrição não deve levar mais do que alguns minutos por idéia. Você precisa capturar apenas a essência da idéia.

Priorização

Em algumas situações, a geração de idéias é o único objetivo, assim, o processo está completo. No entanto, na maioria dos casos, incluindo o workshop de requisitos, é necessário priorizar as idéias que permaneceram após o expurgo. Afinal de contas, nenhuma equipe de desenvolvimento pode fazer "tudo o que todos pensaram". Uma vez que o agrupamento tenha se estabilizado e atingido o consenso, é hora de ir para o próximo passo. Novamente, várias técnicas podem ser usadas; nós iremos descrever duas que usamos rotineiramente.

Resultados da votação acumulativa:
Idéia 1 $380
Idéia 2 $200
Idéia 3 $180
Idéia 4 $140
Idéia 5 ...
.
.
.
Idéia 27 ...

Votação Acumulativa: Teste dos Cem Dólares. Este simples teste é divertido, legal e fácil de fazer. Cada pessoa recebe 100 "notas de idéias" para gastar na "compra de idéias". (Você pode até mesmo adicionar um kit de "notas de idéias" no inventário de cupons do workshop). Cada participante é convidado a escrever em seu bloco de notas, o quanto pagaria para cada idéia. Então, depois que os participantes tiverem tido a chance de votar, o facilitador irá tabular os resultados e fornecer a ordem de classificação. Pode também ser útil fazer um rápido

histograma dos resultados para que os participantes possam ver o impacto visual de suas decisões.

Este processo é extremamente simples e normalmente funciona bem. No entanto, você deve ficar prevenido dos seguintes avisos. Primeiro: isso funcionará bem apenas uma vez. Você não pode usar a mesma técnica duas vezes no projeto, porque uma vez que os resultados sejam conhecidos, os participantes irão influenciar os resultados na próxima rodada. Por exemplo, se você for um participante e a sua característica favorita for o primeiro da lista, mas se a sua segunda característica favorita não estiver numa posição honrosa, você pode colocar todo o seu dinheiro sobre a sua segunda característica. Você está confiante de que os outros eleitores irão ver que a sua característica favorita ainda faz a diferença.

Da mesma forma, você pode achar necessário limitar a quantidade de gastos de uma característica. Caso contrário, um participante trapaceiro conhecendo perfeitamente que "outros itens" tais como "Execução rápida" e "Facilidade de uso" fazem o corte para o topo da lista, pode gastar todo o seu dinheiro sobre "Executar na plataforma Mac" e elevá-la para uma prioridade maior. Por outro lado, você pode desejar a permissão de um limite elevado, contanto que você queira saber onde estarão os grandes votos. Eles podem representar necessidades de alta prioridade de uma comunidade restrita de stakeholders.

A Categorização "Crítica, Importante e Útil". Um colega nos ensinou uma outra técnica que também é bastante efetiva, especialmente com um pequeno grupo de stakeholders ou mesmo com apenas um stakeholder, tal como quando você precisar da opinião do seu chefe de suas prioridades. Nesta técnica, é dado a cada participante um número de votos igual ao número de idéias, mas cada voto deve ser categorizado como "crítico", "importante" ou "útil". O truque nesta técnica é a regra de que cada stakeholder dá apenas um dos três votos de cada categoria; assim, apenas um terço das idéias podem ser consideradas críticas.

- *Crítico* significa "indispensável", sugerindo que um stakeholder não poderia estar apto a usar um sistema sem esta característica. Sem a característica, o sistema não cumpriria a sua missão principal, seu papel e, assim, digno de liberar.
- *Importante* significa que seria uma significativa perda de utilidade para o segmento de cliente, mercado, rendimento ou de serviços aos novos clientes. Se esse item não for implementado, alguns usuários não gostarão do produto e não irão comprá-lo.
- *Útil* significa que seria bom ter. A característica que torna a vida mais fácil, faz o sistema mais apelativo, mais divertido, ou de grande utilidade.

Nota: Com este esquema, todas as idéias que sobreviveram ao processo de expurgo terão ao menos um voto "útil", evitando insultar a quem tenha submetido.

Num grupo grande de participantes, cada item terá um misto de categorias, mas não é realmente um problema. O facilitador tem um truque: Apenas multiplique os votos "críticos" por 9, "importantes" por 3 e "úteis" por 1 e some os resultados! Isto tende a espalhar os resultados pesadamente a favor dos votos

"críticos", e assim, todas as necessidades "críticas" dos stakeholders borbulharão para o topo da lista.

Brainstorming Baseado em Web

Até agora, discutimos um processo para que o brainstorming funcione efetivamente bem quando todos os stakeholders reunidos ao mesmo tempo forem relativamente pró-ativos e não excessivamente inibidos, quando o facilitador for experiente e políticas do stakeholder forem gerenciáveis. Sem dúvida, não existe substituto para o tempo gasto em conjunto pelos desenvolvedores e stakeholders externos. Cada um irá se lembrar dos vários pontos importantes e de assuntos discutidos por outros; perspectivas e respeito mútuo são normalmente os subprodutos do processo. Então, o workshop de requisitos e o brainstorming presencial são, sem dúvida, a nossa abordagem preferida.

Mas, algumas vezes, o brainstorming presencial não é possível. Nessas situações, uma das alternativas é o uso da Internet ou uma intranet para facilitar o processo de brainstorming criando-se um grupo de discussão. Esta técnica pode ser particularmente adaptada para aplicações avançadas de desenvolvimento onde são necessárias pesquisas ou quando uma visão de longo alcance é crítica, o conceito for inicialmente confuso, e uma grande variedade de informações e significante número de usuários e de outros stakeholders estiverem envolvidos.

Com esta técnica, o líder de projeto patrocina um servidor de lista ou página Web para registrar e comentar as características do produto. O registro de idéias e comentários podem ser feitos tanto de forma anônima quanto pela criação de autores com base nas construções criadas pelo administrador. Uma vantagem desta técnica é a persistência; idéias e comentários podem ser circulados por um longo período de tempo, com total registro de todas as ramificações de cada idéia. Talvez a vantagem mais importante e única deste processo esteja no fato de que idéias podem crescer e amadurecer com o passar do tempo.

O Caso de Estudos: O Workshop de Requisitos do HOLIS 2000

Permita-nos retornar ao nosso caso de estudos. Enquanto o processo de entrevista estava em andamento, a equipe de desenvolvimento reuniu-se com o marketing e decidiu realizar um workshop de requisitos para o projeto HOLIS 2000.

Participantes

Depois de pensar sobre o assunto, a equipe decidiu não trazer um facilitador externo, ao invés disso, definiram que Eric, o diretor de marketing, facilitaria o workshop. A equipe também decidiu que haveria a participação de dois membros da equipe de desenvolvimento: Cathy, a gerente de produto e Pete, o gerente de desenvolvimento. A equipe sentiu que tanto Cathy quanto Pete poderiam falar pela equipe e que estavam aptas em contribuir com o conteúdo, ambas como novas proprietárias. Os outros membros da equipe não iriam participar, mas iriam simplesmente assistir o workshop a fim de observar o processo, ouvir os clientes e ver, diretamente, os resultados.

A equipe também decidiu incluir a representação de quatro "classes" de clientes e convidou os seguintes participantes:

1. Distribuidores: John, diretor executivo da maior companhia de distribuição, e Raquel, a gerente geral da companhia de distribuição exclusiva na Europa.
2. David, um construtor de casas local com experiência em comprar e instalar sistemas concorrentes no mercado.
3. Betty, uma empreiteira elétrica local.
4. As perspectivas de proprietários, identificadas com a ajuda de Betty, que passou por um processo de construção, ou considerou a construção de uma residência de última geração.

A seguinte lista fornece maiores detalhes sobre os participantes.

Nome	Papel	Título	Comentários
Eric	Facilitador	Diretor de Marketing	
Cathy	Participante	Gerente de Projetos do HOLIS 2000	Defensor do projeto
Pete	Participante	Gerente de Desenvolvimento de Software	Responsável pelo desenvolvimento do HOLIS 2000
Jennifer	Participante		Perspectiva do proprietário
Elmer	Participante		Perspectiva do proprietário
Gene	Participante		Perspectiva do proprietário
John	Participante	Diretor Executivo da empresa Equipe de Automação	Maior distribuidor da Lumenations
Raquel	Participante	Gerente da EuroControls	Distribuidor Europeu da Lumenations
Betty	Participante	Presidente da Krystel Eletric	Empreiteira Elétrica local
David	Participante	Presidente da Rosewind Construction	Construtor de casas personalizadas
Vários membros	Observador	Equipe de desenvolvimento	Todos os membros da equipe que estiverem disponíveis

O Workshop

Antes do workshop, a equipe disponibilizou um pacote de aquecimento contendo:

- Um artigo recente sobre tendências de iluminação em automação de casas
- Cópias de entrevistas seletivas que haviam sido conduzidas
- Uma lista sumarizada das necessidades que foram identificadas até a data

Eric preparou suas habilidades de facilitador, e Cathy trabalhou na logística do workshop.

A sessão

A sessão foi realizada num hotel próximo ao aeroporto e começou prontamente às 8:00 horas. Eric apresentou a agenda do dia e as regras do workshop, incluindo os cupons de workshop. A Figura 11–2 fornece uma visão do workshop.

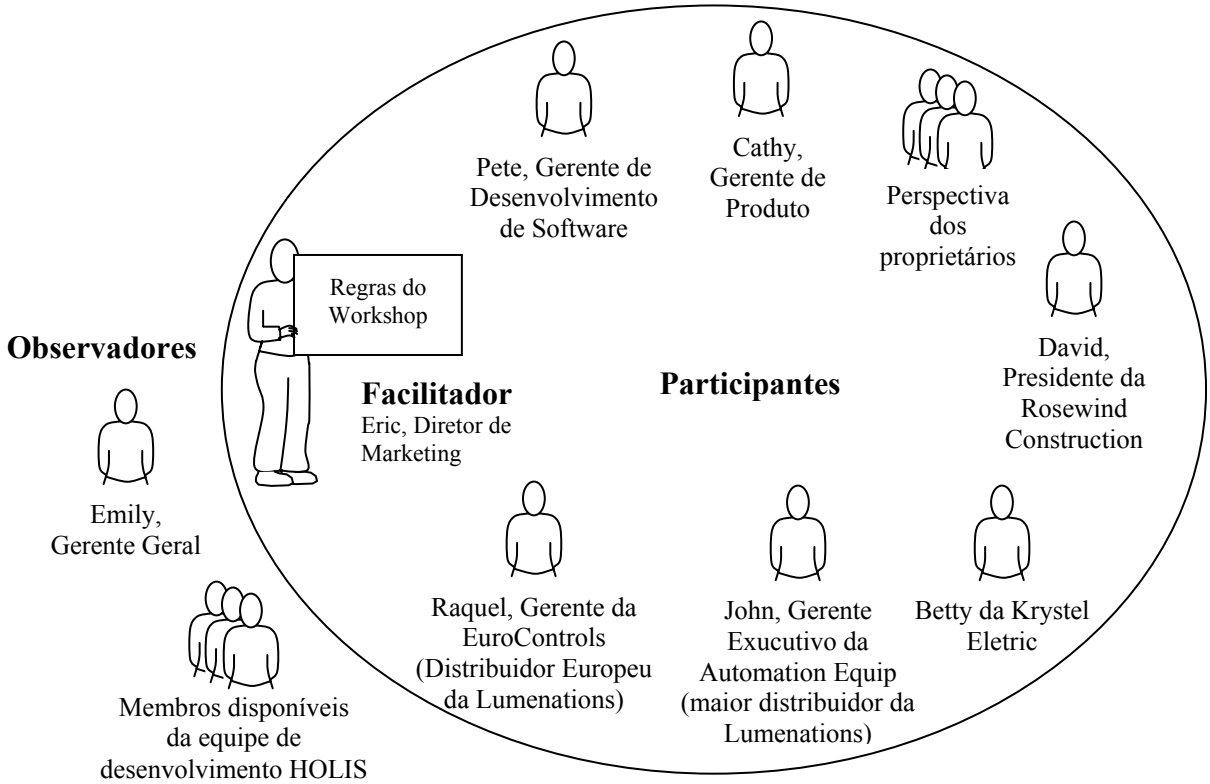

Pete, Gerente de Desenvolvimento de Software
Cathy, Gerente de Produto
Perspectiva dos proprietários
Regras do Workshop
Observadores
Facilitador
Eric, Diretor de Marketing
Participantes
David, Presidente da Rosewind Construction
Emily, Gerente Geral
Raquel, Gerente da EuroControls (Distribuidor Europeu da Lumenations)
John, Gerente Exucutivo da Automation Equip (maior distribuidor da Lumenations)
Betty da Krystel Eletric
Membros disponíveis da equipe de desenvolvimento HOLIS

Figura 11–2 Estrutura do workshop de requisitos do HOLIS 2000

Em geral o workshop caminhou muito bem, e todos os participantes estavam preparados para dar suas informações prontamente. Eric fez um bom trabalho de facilitar a reunião, mas uma situação desagradável ocorreu quando Eric argumentou com a Cathy sobre as prioridades de duas características. (A equipe decidiu que nos futuros workshops, um facilitador externo seria contratado). Eric conduziu uma sessão de brainstorming sobre as características potenciais do

HOLIS, e a equipe usou a votação acumulativa para decidir as prioridades relativas. Os resultados são apresentados na Tabela 11–1.

Tabela 11–1 Características do workshop HOLIS, ordenados por prioridade

ID	Características	Votos
23	Cenas de iluminação personalizadas	121
16	Configuração do tempo automático de iluminação, etc.	107
4	Características de segurança pré-definidas, p.ex., lâmpadas e alarmes	105
6	100% de confiabilidade	90
8	Fácil de programar, unidade de controle não-PC	88
1	Facilidade de programar as estações de controle	77
5	Configuração de ausências	77
13	Toda lâmpada pode ser apagada	74
9	Usar meu próprio PC para programar	73
14	Características de entretenimento	66
20	Portas da garagem fechadas	66
19	Ligar automaticamente as lâmpadas quando a porta for aberta	55
3	Interface com o sistema de segurança residencial	52
2	Facilidade para instalar	50
18	Ligar automaticamente as luzes quando alguém bate a porta	50
7	Ligar e desligar instantâneo	44
11	Poder dirigir cortinas, sombras, bomba d'água e motores	44
15	Controlar a iluminação via telefone	44
10	Interface com o sistema de automação residencial	43
22	Modo gradual: aumento ou redução da luminosidade lentamente	34
26	Estação de controle principal	31
12	Facilidade de expansão quando remodelado	25
25	Interface de usuário internacionalizado	24
21	Interface com o sistema de áudio/vídeo	23
24	Restauração após falha no fornecimento de energia elétrica	23
17	Controle HVAC	22
28	Ativação via voz	7
27	Apresentação no web site do usuário	4

Análise de Resultados

Os resultados do processo apresentaram-se como esperado, exceto por dois itens significativos:

1. "Segurança pré-definida" surgiu na lista com alta prioridade. Esta característica havia sido mencionada em entrevistas anteriores, mas não foi colocada na lista de prioridades de qualquer entrevistado. Depois de uma rápida revisão, Cathy notou que a segurança pré-definida, tais como a habilidade de flash de luz, a sirene opcional e

sistemas de chamada para emergências, aparentemente não eram fornecidos pelos sistemas da concorrência. Os participantes comentaram que embora tenha sido uma surpresa esta informação, eles acham que isso deve ser uma diferenciação competitiva e, de acordo com isso, ser uma característica de alta prioridade. Krys e David concordaram. Com base nessa conclusão, o marketing decidiu incluir esta funcionalidade e colocá-la como o seu único diferencial competitivo no mercado. Esta será uma das *características que definem* o HOLIS.

2. Além disso, a característica 25, "Internacionalização da interface do usuário" não recebeu muitos votos. (Isto pareceu fazer sentido à equipe, porque os proprietários localizados nos Estados Unidos não podem se preocupar com o como o produto será vendido na Europa!). O distribuidor, no entanto, disse sem rodeios de que se o produto não estiver internacionalizado desde a versão 1.0, ele não seria introduzido na Europa. A equipe anotou esta posição e concordou em explorar o esforço necessário para alcançar a internacionalização na release 1.0[3]

[3] Este assunto demonstra que um dos problemas com a votação acumulativa. Nem todos os stakeholders são criados igualmente. Falhas em atingir a internacionalização, que não havia aparecido no "visor do radar" da equipe antes do workshop, poderia se tornar um equívoco de requisitos estratégicos de proporções significativas.

Capítulo 12

Storyboarding

Pontos chaves

- O propósito do storyboarding é elucidar as reações iniciais "Sim, mas".
- Os storyboards podem ser passivos, ativos ou interativos.
- Os storyboards identificam os jogadores, explicam o que acontece com eles e descreve como acontece.
- Fazer com que o esboço do storyboard seja fácil de modificar e não disponível.
- Os storyboards iniciais são comuns em todos os projetos que tenham conteúdos novos e inovadores.

Talvez nenhuma outra técnica de elucidação tenha recebido tantas interpretações quanto o "storyboarding". Apesar disso, várias dessas interpretações concordam com o propósito do storyboarding, que é o de descobrir as reações iniciais dos usuários sobre o conceito proposto pela aplicação. Ao fazer isso, storyboards oferecem uma das técnicas mais efetivas de atacar a síndrome "Sim, mas". Com o storyboarding, as reações dos usuários podem ser observadas logo no início do ciclo de vida, bem antes que os conceitos tenham comprometido o código e, em muitos casos, até antes que as especificações detalhadas tenham sido desenvolvidas. Os especialistas em fatores humanos, por anos, nos têm dito que o poder dos storyboards não deve ser subestimado. De fato, a indústria do cinema tem usado esta técnica desde que a primeira centelha de luz foi projetada na tela.

Os storyboarding efetivos aplicam ferramentas que são baratas e fáceis de usar. O storyboarding:

- É extremamente barato.
- É amigável, informal e interativo.
- Fornece uma revisão inicial das interfaces de usuário do sistema.
- É fácil de criar e modificar.

Os storyboards são também uma forma poderosa de facilitar a "síndrome da página-branca". Quando os usuários não sabem o que querem, até um storyboard muito pobre é bom para elucidar uma resposta como "Não, isto não é o que eu queria, deveria ser o seguinte...", e o jogo começa.

Storyboards podem ser usados para acelerar o desenvolvimento conceitual de várias diferentes facetas de uma aplicação. Storyboards podem ser usados para entender a visualização de dados, para definir e entender as regras de negócio que serão implementadas numa nova aplicação de negócio, para definir algoritmos e outras construções matemáticas que serão executados dentro de um sistema embutido, ou para demonstrar relatórios e outras saídas impressas da revisão inicial. De fato, os storyboards podem e devem ser usados para, virtualmente,

qualquer tipo de aplicação onde conhecer as reações iniciais de usuários seja um dos fatores chaves de sucesso.

Tipos de Storyboards

Basicamente, um storyboard pode ser qualquer coisa que a equipe quer que seja, e a equipe deve se sentir livre para usar sua imaginação para pensar num storyboard de aplicações específicas. Storyboards podem ser categorizados em três tipos, dependendo do modo de interação com o usuário: passivo, ativo ou interativo.

- *Storyboards passivos* contam uma estória ao usuário. Podem ser consistidos de esboços, figuras, "screen shots", apresentações PowerPoint, ou amostras de saídas. Num storyboard passivo, o analista interpreta o papel do sistema e simplesmente conduz o usuário através do storyboard, com uma exposição "Quando você fizer isto, isso acontece".
- *Storyboards ativos* tentam fazer o usuário ver "um filme que não foi produzido ainda". Storyboards ativos são animados ou automatizados, talvez por uma ferramenta de apresentação seqüencial de slides ou de animação, ou até por um filme. Storyboards ativos fornecem uma descrição automatizada sobre a maneira de como o sistema de comporta num cenário normal de uso ou de operação.
- *Storyboards interativos* permitem que o usuário experimente o sistema tanto de maneira realística quanto prática. Sua execução necessita da participação do usuário. Storyboards interativos podem ser simulações, maquetes ou ser tão sofisticado quanto um código descartável. Um storyboards sofisticado construído com códigos descartáveis pode estar muito próximo a um **protótipo descartável** (discutido no próximo capítulo).

Como ilustra a Figura 12–1, estas três técnicas de storyboarding oferecem um continuum de possibilidades, variando desde um exemplo das saídas até demonstrações interativas reais.

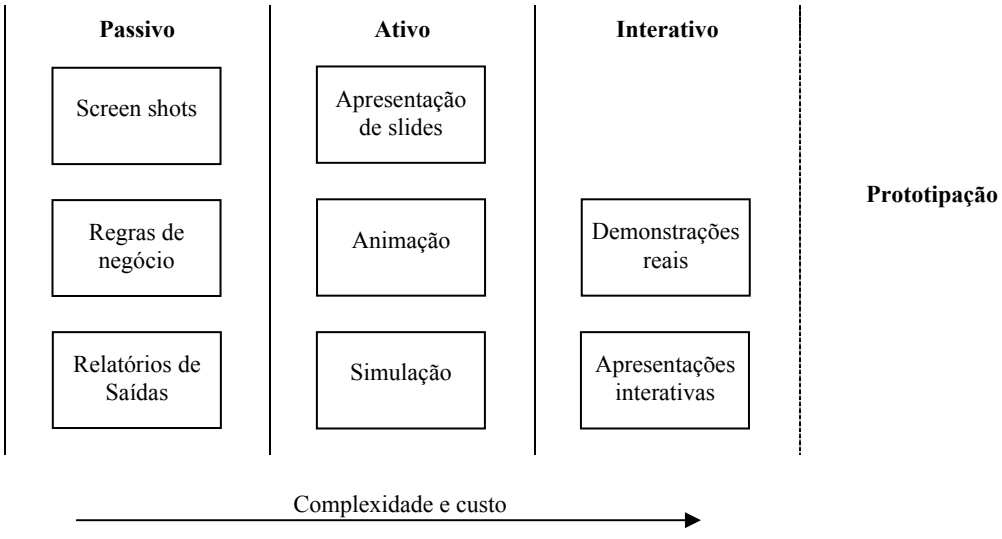


Figura 12–1 Regras do brainstorming

De fato, o limite entre storyboards sofisticados e protótipos iniciais de um produto é algo nebuloso.

A escolha da técnica de storyboarding varia de acordo com a complexidade do sistema e dos riscos de enganos sobre o que o sistema precisa fazer.

Um sistema sem precedentes e inovador que tenha uma definição leve e confusa pode necessitar de múltiplos storyboards, partindo do passivo para o interativo conforme o entendimento da equipe sobre os aperfeiçoamentos do sistema.

O que os Storyboards Fazem

A Branca de Neve e os Sete Anões da Disney, o primeiro filme animado então produzido, usou storyboards, e eram rotineiramente usados como parte integrante do processo criativo nos filmes e desenhos animados. Virtualmente, todos os filmes, com características animadas ou desenhos animados começam com storyboards. Eles representam informações cruas de criação usadas no desenvolvimento de personagens e no enredo da estória.

Em software, storyboards são usados com maior freqüência para trabalhar detalhes de interfaces homem-máquina. Nesta área, geralmente uma das mais voláteis, cada usuário provavelmente possui diferentes opiniões sobre como a interface deve funcionar. Storyboards para sistemas baseados em usuários lida com três elementos essenciais de qualquer atividade:

1. Quem são os jogadores
2. O que acontece com eles
3. Como acontece

O elemento *quem* define os jogadores, ou usuários do sistema. Num sistema de software, como discutimos anteriormente, o "quem" são jogadores tais como usuários e outros sistemas ou periféricos – os atores que interagem com a solução sistêmica que estamos construindo. Para usuários, a interação é tipicamente descrita via informações de telas ou de formulários de entrada de dados, dados e relatórios de saída, ou via outros tipos de periféricos de entrada e saída, tais como botões, chaves, displays e monitores. Para periféricos e sistemas, interações são realizadas via uma interface de software ou hardware, tais como protocolo de comunicação ou sinais do drive de controle de motor.

O elemento *o que* representa o comportamento de como os usuários interagem com o sistema ou, alternativamente, o comportamento de como o sistema interage com o usuário. O elemento *como* representa o estado que o jogador ou o sistema assume durante a interação.

Por exemplo, nós fizemos um storyboard para um veículo de entretenimento automático de passeio no parque.

- O *quem* representa o passageiro que passeia com o veículo.
- O *o que* representa o comportamento do veículo quando vários eventos são fornecidos pelo passageiro.
- O *como* fornece futuras descrições de como essa interação acontecem – eventos, transições de estados – e descreve tanto o estado do passageiro

(surpresa, assustado) quanto o estado do veículo (acelerando, freando, descarregando)

Ferramentas e Técnicas para o Storyboarding

As ferramentas e técnicas para o storyboarding podem ser tão variadas quanto a imaginação dos membros da equipe e dos usuários do sistema. A construção de storyboarding passivos tem sido realizada com ferramentas tão simples quanto papel e lápis ou notas Post-It. Storyboarding passivos sofisticados podem ser construídos com gerenciadores de apresentação tais como o PowerPoint ou Harvard Business Graphics. Storyboards interativos com usuários ativos ou passivos têm sido construídos com HyperCard, SuperCard e com a utilização de vários pacotes que permitem rápido desenvolvimento das telas de usuários e relatórios de saída. Storyboards interativos podem ser construídos com uma variedade de pacotes de software específicos para prototipação interativa, tal como Dan Bricklin's Demo It. Ferramentas tais como Director da Macromedia e Cinemation da Vividus Corp. podem ser usados para criar animações e simulações mais complexas.

Num exemplo simples, ocorrido na RELA, Inc, um membro da equipe também tinha interesse em cartooning. No estágio de conceito de um projeto, ele apenas esboçou meia dúzia de desenhos simples que mostravam os vários aspectos da interface do produto em seu uso típico. Esta foi uma maneira simples e barata de provocar uma reação dos potenciais usuários. Além disso, a natureza dos cartoons evitou alguns dos potenciais problemas do storyboards, como veremos mais tarde. Infelizmente, nenhum outro cartunista foi encontrado depois que ele saiu da empresa, deixando-nos a procurar técnicas alternativas de sotryboarding!

Em nosso atual esforço, cujo foco na maioria das vezes são as aplicações ISV, nós nos viramos muito bem usando o PowerPoint ou outros gerenciadores de apresentação de desktop comum, em combinação com amostrar de screen shots construídos com as mesmas ferramentas usadas para construir GUIs (Graphical User Interface) de aplicações. Interessantemente, o incrível avanço na técnica de storyboarding pode muito bem ter sido a adição da capacidade de animação ao PowerPoint. Repentinamente, a nossa habilidade de expressar a dinâmica e interatividade aumentara enormemente.

Dicas do Storyboarding

O storyboarding é uma técnica projetada para obter um feedback inicial do usuário usando ferramentas inexpressivas. Assim, storyboards são particularmente efetivos em atacar a síndrome "Sim, mas". Eles ajudam também a atacar a síndrome das "Ruínas Desconhecidas" elucidando imediatamente o feedback de usuários tal como o que o sistema "parece que não deve fazer". Mas, como em qualquer técnica, certas advertências de aplicam. Aqui estão algumas dicas que devemos ter em mente quando se pratica a técnica do storyboarding.

- *Não invista muito tempo no storyboarding*. Clientes ficarão intimidados a fazer mudanças se o storyboard se parecer muito com o produto real ou eles podem pensar que estão insultando você, um problema particularmente difícil em algumas culturas. Tudo bem; mantenha o

storyboard deselegante e grosseiro, até mesmo cru. (Veja a estória do storyboarding no final deste capítulo).

- *Se você não fizer nenhuma mudança, você não aprenderá qualquer coisa*. Faça o storyboard para que seja fácil de modificar. Você deve estar apto a modificar um storyboard em algumas horas.
- *Não faça o storyboard muito bom*. Se você fizer isso, os clientes irão querer levá-lo. (Num projeto real, nós sofremos por anos tendo que dar suporte num produto Excel/VB que nunca pretendeu ser mais do que um storyboard). Mantenha o storyboard como rascunho; use ferramentas e técnicas que não causem perigo neste campo, especialmente para storyboards que são codificados. (*Dica*: Se a aplicação será implementada em Java, escreva o storyboard em VB).
- *Sempre que possível, faça o storyboard interativo*. A experiência do cliente ao usar a aplicação irá gerar maior feedback e permitir descobrir mais requisitos novos do que o storyboard passivo.

Sumário

Neste capítulo, nós aprendemos sobre uma técnica muito simples e barata para elucidar requisitos. De ser forma, um storyboard é qualquer coisa que você pode construir rapidamente e economicamente que elucide uma reação "Sim, mas" do usuário.

Nós podemos afirmar com certeza que nunca deixamos de aprender alguma coisa com o storyboard, e nunca houve um caso em que nós tenhamos saído de um storyboard com exatamente o mesmo conhecimento que tínhamos antes de entrarmos. Assim, nosso aviso para a equipe de desenvolvimento é:

- Storyboard no início.
- Storyboard com freqüência.
- Storyboard em todos os projetos que possuam conteúdos novos e inovadores.

Ao fazer isso, você irá ouvir o quanto antes o "Sim, mas", o qual irá lhe ajudar a construir sistemas que melhor atenda as necessidades dos usuários reais. E, talvez, você irá rapidamente e economicamente ver seu trabalho realizado!

Uma Estória de Storyboarding

(Alguns fatos foram alterados para proteger inocentes e culpados desta quase verdadeira estória). A estória ocorreu durante o desenvolvimento de um periférico eletromecânico complexo para um hospital farmácia. O cliente era um fabricante da Fortune 1000; o fornecedor, a nossa empresa, havia sido contratada para desenvolver este novo produto, um sistema eletromecânico óptico complexo para controle de fluídos. O projeto estava com problemas.

Um dia, o chefe dos gerentes do projeto contratado (o chamaremos de "autor") recebeu a seguinte chamada telefônica da gerência superior do cliente (um vice-

presidente Sênior, "Sr. Big", uma pessoa poderosa quem nós nunca havíamos conhecido antes.

Sr. Big: *Autor, como anda o nosso projeto favorito?*
Autor: *Não muito bem.*
Sr. Big: *Estou ouvindo direito? Tudo bem, não existe problema grande o suficiente que não possa ser resolvido. Reúna sua equipe evenha até aqui para uma reunião. Que tal na quarta-feira?*
Autor: (afobadamente folheando a agenda da equipe para quarta-feira*) Quarta-feira está perfeito.*
Sr. Big: *Excelente. Venha e traga toda a sua equipe. Outra coisa, não se preocupe com os custos de viagem. Nós cobriremos. Pro inferno! Compre aquelas passagens só de ida.*
Autor: (Engolindo seco) *Está bem, obrigado. Nós nos veremos na quarta.*

No dia indicado, nós entramos num grande salão de conferência com a equipe de projeto do cliente todos sentados na extremidade remota. A equipe claramente estava reunida já algum tempo. (Questão: Por que a equipe sentiu necessidade de se reunir antes que a verdadeira reunião começasse?). O Autor, que não é um "marinheiro de primeira viagem", caminhou para a outra extremidade da sala e sentou-se próximo ao Sr. Big (a teoria diz que será difícil para o Sr. Big gritar com o Autor se eles estiver próximo ao Sr. Big; além disso, se ele golpear o Autor, existe a chance de vencê-lo na justiça e recuperar os lucros de projeto!).

Depois de uma breve discussão, o Autor notou que dentre os vários problemas importantes que preocupam o projeto, o problema da "não convergência dos requisitos" estava causando atrasos e excedendo os custos. O Sr. Big disse, "*Dê-me um exemplo*". O Autor lhe deu um exemplo. Os membros da equipe do cliente imediatamente começaram a argumentar entre si, talvez demonstrando que este era o real problema. O subcontratado sussurrou um pequeno sinal de alívio. O Sr. Big observou a equipe por um momento e então disse, "*Muito engraçado. Dê-me mais um exemplo*". A equipe do Autor arrancou cinco impressões coloridas, cada uma realizada com perfeição profissional, do painel frontal proposto e disse: "*nós apresentamos todas estas opções de projetos semanas atrás, e não conseguimos convergir para nenhum desses projetos, e estamos exatamente no momento em que precisamos disso*". O Sr. Big disse, "*Isso não pode ser tão difícil. Equipe, escolha um*". Os membros da equipe do cliente então brigaram entre si novamente. O dia passou desse jeito. Não houve convergência. Havia pouca esperança.

Na manhã seguinte, o Autor foi convidado para um café da manhã com um membro da equipe de projeto (Membro da Equipe). O Membro da Equipe, também um costureiro, arrancou o fundo de feltro de um pano, cortou com a tesoura, e depois de colori-lo disse "*Eu gostaria de facilitar a parte da interface do usuário na reunião usando essas ferramentas*".

Autor: *"Você está brincando; não tem como fazer isso funcionar. Isso parecerá tolo e anti-profissional".*
Membro da Equipe: *"Eu entendo, mas quão efetivo fomos ontem?"*

O Autor, politicamente correto, não disse a primeira palavra que lhe veio à mente. A segunda palavra foi "*OK*".

O próximo dia, o tom na sala de reuniões foi muito diferente. A equipe do cliente tinha, novamente, chegado cedo, mas desta vez estavam silenciosos e carrancudos ao invés de descomedido e excitado. (*Análise: Agora eles sabem o quanto nós estávamos atados e sem respostas. Eles haviam planejado nos matar, mas agora sabem que estávamos sendo injustiçados*).

Para iniciar a reunião, o Membro da Equipe colocou uma peça de feltro de 3' por 5' (1 × 2 m) sobre a parede, gerando risos, mas não desinteresse por parte do cliente.

O Membro da Equipe colocou grandes interruptores para chaves de potência e vários botões de modo-terapia feitos de feltro, sobre o painel frontal e disse "*Agora este projeto funciona?*".

O cliente olhou para a parede e disse "*Não, mas por que você não desloca o botão de parede de emergência para baixo?*".

O Membro da Equipe disse: "*Aqui, por que você não faz isso?*", e passou uma tesoura para o cliente.

O cliente pegou a tesoura, e o Membro da Equipe deixou a sala. O cliente continuou a sessão de projeto interativo com feltros e tesouras. Uma hora mais tarde, o cliente olhou a parede e disse: "*Está muito bom; construa-o*".

Permita-nos descobrir a moral da estória com uma pequena leitura de pergunta e resposta:

Pergunta: Por que a loucura do feltro funcionou e a impressão colorida não?
Resposta: Existem duas razões.

- *Interatividade*: O que o cliente pode fazer com cinco desenhos, dos quais apenas uma parte é satisfatória?
- *Usabilidade*: Quão assustador pode ser cortar um grande pedaço de feltro?

O cliente, que tinha o domínio da especialidade, mas não necessariamente o projeto, projetou uma solução adequada para o seu próprio problema.

Nós adotamos a casa de feltro como nossa e fixamos na parede como uma constante lembrança sobre o que nós aprendemos. A interface do usuário, embora não tenha sido uma solução ótima, nunca mudou e se adequou aos propósitos dos interessados. Mas, infelizmente, o projeto não foi um grande sucesso de venda, embora o produto tenha ido para o mercado e alcançado sucesso. Como dissemos anteriormente, este foi apenas um dos problemas enfrentados neste particular projeto.

- **Lição 1:** Entender as necessidades do usuário é um problema frágil e confuso. Utilize ferramentas leves e malucas – storyboards e feltros, se necessário – para atacá-lo.
- **Lição 2:** A tecnologia é difícil. Pense duas vezes antes de você iniciar um negócio de terceirização de periféricos médicos.

Capítulo 13

Aplicando Use Cases

Pontos chaves

- Os use cases, como storyboards, identificam quem, o que e como do comportamento do sistema.
- Use cases descrevem as interações entre um usuário e um sistema, concentrando-se no que o sistema "faz" para o usuário.
- O modelo use-case descreve a totalidade do comportamento funcional do sistema.

No Capítulo 12, descrevemos o storyboarding e discutimos como você pode usar storyboards para mostrar *quem*, *o que* e *como* do comportamento do sistema e do usuário. Use cases são uma outra técnica para expressar este comportamento. Nós introduzimos brevemente esta técnica no Capítulo 2 e 5, onde usamos para nos ajudar a modelar o comportamento de um negócio.

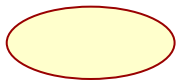
Neste capítulo, nós iremos desenvolver a técnica use-case descrevendo como podemos usá-los para entender o comportamento do sistema que estamos desenvolvendo, em oposição ao entendimento do comportamento do negócio que irá utilizar o sistema. Em outras palavras, usaremos use cases como uma técnica de elucidação para entender o comportamento necessário da aplicação que iremos desenvolver para resolver o problema do usuário. Use cases são uma técnica tão importante para capturar e especificar os requisitos do sistema que nós iremos desenvolvê-lo na Habilidade de Equipe 5, Refinando a Definição do Sistema e Habilidade de Equipe 6, Construindo o Sistema Correto.

A técnica use-case é parte integral do método da Engenharia de Software Orientado a Objetos, como descrito no livro *Object-Oriented Software Engineering, A Use Case Driven Approach* (Jacobson et al. 1992). Este método de análise e projeto de sistemas complexos é "dirigido por use case", uma maneira de descrever o comportamento do sistema a partir da perspectiva de como os vários usuários interagem com o sistema para atingir seus objetivos. Esta abordagem, centrada no usuário, fornece a oportunidade para explorar o comportamento do sistema com o envolvimento do usuário desde o início.

Além disso, como mencionamos anteriormente, use cases servem como representações UML para requisitos de um sistema. Além de capturar os requisitos do sistema, os use cases desenvolvidos no processo de elucidação de requisitos irão ser muito úteis durante as atividades de análise e projeto. De fato, o método use-case é importante durante todo o ciclo de vida do software; como exemplo, os use cases podem assumir um papel significativo durante o processo de testes. Nos capítulos seguintes iremos desenvolver a técnica use-case com maior profundidade, por agora, nós precisamos entender apenas sobre como aplicar use cases para capturar os requisitos iniciais do sistema.

Nós começamos com uma definição menos formal do que iremos fornecer posteriormente:

> *Um use case descreve uma seqüência de ações que um sistema executa para produzir um resultado de valor a um particular ator.*



Em outras palavras, use cases descrevem as interações entre um usuário e um sistema, e eles focam sobre o que o sistema "faz" para o usuário. Além disso, como as ações são descritas numa seqüência, é fácil "seguir as ações" e entender o que o sistema faz para o usuário. Em UML, o use case é representado por uma simples figura oval com um nome abaixo.

Na elucidação de requisitos, use cases podem elucidar e capturar requisitos do sistema. Cada use case descreve uma série de eventos nos quais um particular ator, tal como a "Jenny a Modelo", interage com o sistema, tal como o "Sistema de Agendamento de Clientes da Agência de Modelos Ad Lib", para atingir um resultado de valor para a Jenny, tal como a localização do próximo trabalho de desfile.

Construindo o Modelo Use-Case

O *modelo use-case* de um sistema consiste de todos os atores do sistema e de todos os use cases com os quais os atores interagem no sistema. O modelo use-case descreve a totalidade do comportamento funcional do sistema. O modelo use-case também mostra os relacionamentos entre use cases, os quais facilitam nosso entendimento do sistema.

O primeiro passo na modelagem use-case é criar um diagrama do sistema que descreva a fronteira do sistema e identifique os atores do sistema. Isso apresenta um belo paralelo com os passos 3 e 4 dos cinco passos da análise do problema, onde nós identificamos os stakeholders do sistema e definimos a fronteira do sistema. Nós também aprendemos nos Capítulos 4, 5 e 6, como identificar os atores que irão interagir com o sistema.

Por exemplo, num sistema de gerenciamento de estoques (Jacobson et al. 1992), a fronteira do sistema pode se parecer como a Figura 13–1.

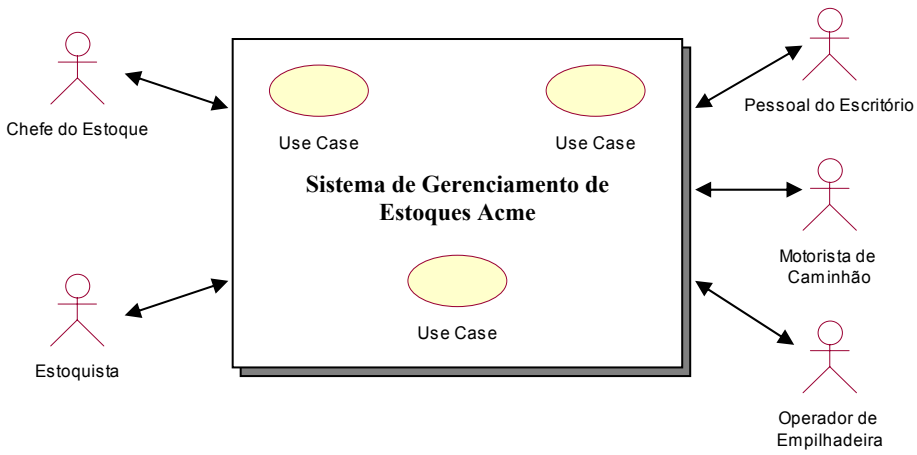


Figura 13–1 O sistema de estoque inicial, com atores identificados

Você pode ver que o sistema é usado por alguns usuários, cada um dos quais interagem com o sistema para tingir um objetivo operacional específico.

A análise futura do sistema determina que certas linhas de comportamento do sistema são necessárias para suportar as necessidades dos usuários. Essas linhas são os use cases, ou a seqüência específica pelas quais os usuários interagem com o sistema para realizar um objetivo específico. Exemplos desses use cases de sistema podem incluir:

- Distribuição manual dos itens dentro de estoque.
- Inserção de um novo item no estoque.
- Verificar itens de estoque.

Aplicando Use Cases para Elucidação de Requisitos

A noção de use cases pode ser descrita sob a perspectiva do usuário do sistema de forma muito simples. Use cases são descritos em linguagem natural. São fáceis de descrever e documentar. Isto fornece um formato estruturado simples, ao redor do qual a equipe de desenvolvimento e os usuários podem, juntos, trabalhar para descrever o comportamento de um sistema existente ou para definir o comportamento de um novo sistema. E, é claro, cada usuário individualmente irá, naturalmente, se concentrar nas capacidades do sistema que serão necessárias para melhor fazer o seu trabalho. Se, além disso, o comportamento do sistema for totalmente explorado com todos os potenciais usuários, a equipe terá realizado um grande avanço em direção aos objetivos de entender por completo o comportamento desejado do sistema. No final do processo, existirão muito poucas *ruínas de funcionalidade desconhecidas*.

Além disso, na medida em que o método use-case explora as interfaces de usuários diretamente, feedbacks iniciais podem ser obtidos sobre este importante e volátil aspecto da especificação e projeto do sistema.

No entanto, devemos também entender que os usuários do sistema, embora pertençam a uma classe importante, representam apenas uma das classes de stakeholders. Podemos precisar aplicar outras técnicas de elucidação para obter os requisitos de outros stakeholders, tais como clientes não usuários, gerentes, subcontratados, entre outros. Mais ainda, use cases não são tão úteis na identificação de aspectos não-funcionais dos requisitos do sistema, tais como requisitos de usabilidade, confiabilidade, performance, entre outros. Nós iremos contar com outras técnicas para atacar estes assuntos.

Depois que todos os use cases, atores e objetos do sistema tiverem sido identificados, o próximo passo é promover o refinamento dos detalhes de comportamento funcional de cada use case. Essas especificações use-case consistem de descrições textuais e gráficas do use case, escritos do ponto de vista do usuário.

As especificações use-cases podem se entendidas como um repositório que descreve uma série de eventos relacionados, que por sua vez podem ser usados para deduzir outros requisitos que serão desenvolvidos mais tarde. Assim, uma

especificação use case pode incluir o passo "O técnico de manutenção entra com o seu nome (16 caracteres no máximo), sobrenome, entre outros".

Como os use cases definem as interações usuário/sistema, pode ser a hora apropriada de definir, ao menos em conceito, as telas, displays, painéis frontais, entre outras coisas com as quais o usuário interage. Se um sistema de janelas é utilizado para apresentar as informações, pode ser apropriado fazer uma descrição gráfica de alto-nível dos dados a serem exibidos; os detalhes de projeto da interface gráfica formal (GUI), tais como definição de dados cores e fontes, devem ser deixados para as fases posteriores. A Figura 13–2 ilustra uma porção exemplo de uma especificação use-case.

Use Case: Redistribuição dos Itens de Estoque

1. O chefe do estoque dá um comando para redistribuição do estoque.
2. A janela na Figura xxx é apresentada ao chefe do estoque.
3. Os itens podem ser ordenados de várias formas. A ordem é indicada com a seleção do menu Ordenar:
   - Ordem alfabética
   - Ordem por índice
   - Ordem de armazenamento
4. Na tabela "Lugar"; podemos optar em ver, ou todos os Lugares do atual estoque ou, se selecionado um item, os lugares onde esse item existe.

Figura 13–2 Especificação use-case para a distribuição manual do estoque.

Caso de Estudos: Os Use Cases do HOLIS

Impressionado com o poder dos use cases, a equipe de desenvolvimento do HOLIS decidiu usar esta técnica para descrever a funcionalidade do sistema HOLIS em alto-nível. A fim de fazer isso, a equipe organizou uma sessão de brainstorming para definir os use cases significantes os quais serão desenvolvidos mais tarde nas atividades posteriores. Esta "análise do modelo use-case" identificou 20 use cases, alguns dos quais são os seguintes:

Nome	Descrição	Atores
Criar Cena de Iluminação Personalizada	Residentes criam uma cena de iluminação personalizada	Residente, Lâmpada
Iniciar Emergência	Residentes iniciam ação de emergência	Residência
Controlar Iluminação	Residentes ligam ou desligam as lâmpadas, ou reduzem a intensidade desejada de luz	Residentes, Lâmpadas
Trocar Programação	Mudar ou configurar as ações para um particular botão/interruptor	Proprietário / programador
Programar Remotamente	O provedor de serviços de iluminação realiza remotamente a programação com base nas solicitações do residente	Serviços de Iluminação
Tirar Férias	Proprietários configuram a ausência por um longo período	Proprietário / programador
Configurar a Seqüência de Tempo	O proprietário programa a seqüência de iluminação automática com base no tempo	Proprietário / programador

Sumário

Use cases fornecem uma notação estruturada e razoavelmente formal para capturar um subconjunto mais importante das informações de requisitos: como o sistema interage com o usuário para liberar sua funcionalidade. Em muitas aplicações, este subconjunto representa a principal carga de trabalho, tal que use cases podem ser aplicados para expressar os principais requisitos do sistema. Cada use case identificado define os comportamentos necessários do sistema sob a perspectiva de uma classe particular de usuário. Como tal, a técnica é muito útil na elucidação das necessidades do usuário e ajuda a equipe de desenvolvimento representar tais necessidades de maneira que seja prontamente compreensível pelo usuário.

Além disso, como os use cases podem ser usados, posteriormente, nos processos de projeto e testes; eles fornecem uma representação consistente e uma linha consistente através das atividades de requisitos, análise, projeto e testes. Desta forma, a técnica constrói antecipadamente recursos de projeto reutilizáveis que ajudam a aperfeiçoar a eficiência global do processo de desenvolvimento de software. Mais ainda, com a consistência da representação e o suporte fornecido pela UML e por diversas ferramentas de desenvolvimento de aplicações, use cases podem ajudar na automação de vários elementos da atividade de gerenciamento de requisitos. Por estas razões, use cases são notações tão importantes que nós os aplicamos deste ponto em diante como parte integrante das atividades de gerenciamento de requisitos de equipe.

Capítulo 14

Role Playing

Pontos chaves

- O role playing permite que a equipe de desenvolvimento experimente o mundo do usuário a partir da perspectiva do usuário.
- Um roteiro de acompanhamento pode substituir o role playing em algumas situações, com o script se tornando num storyboard vivo.
- Os cartões CRC (Classe-Responsabilidade-Colaboração) freqüentemente usados na análise orientada a objetos, são um derivado do role playing.

Até agora, na Habilidade de Equipe 2, temos discutido uma variedade de técnicas para entender as necessidades dos stakeholders com respeito ao novo sistema que estamos construindo. Nós *falamos* "cara-a-cara" sobre o sistema (entrevista); nós *discutimos* em grupo sobre o sistema (workshops); nós *apresentamos* nossas idéias sobre o sistema (storyboard); e nós *analisamos* como os atores interagem com o sistema (use cases). Todas essas técnicas são boas, elas definem uma estrutura para a nossa compreensão. Mas, vamos admitir, nós não *experimentamos* o sistema.

Neste capítulo, nós discutimos o role playing, o qual permite a equipe de desenvolvimento experimentar diretamente o mundo do usuário pela interpretação do papel do usuário. O conceito por detrás do role playing é muito simples. Embora seja verdade que a observação e o questionamento auxiliam o entendimento, *seria ingenuidade assumir que, apenas através da observação, o desenvolvedor/analista possa chegar à verdade, entendendo em profundidade o problema a ser resolvido ou, através disso, entender claramente os requisitos do sistema que deve solucionar o problema*.

Esta é uma das causas primárias do problema "Sim, mas". Como os sociólogos nos ensinam, todos nós vemos o mundo através de nosso único filtro conceitual. É impossível separar as nossas experiências de vida e influências culturais das observações que fazemos. Por exemplo, podemos observar uma outra cultura realizando uma cerimônia ritualística quando quisermos, mas, provavelmente, será impossível entendermos o que eles significam! O que isso significa para a nossa busca por entender requisitos?

- Devemos entender que muitos usuários não conseguem articular os procedimentos que eles realizam ou as necessidades que devem ser atendidas. Mesmo assim o trabalho é realizado e eles nunca foram questionados sobre isso antes. Além disso, é mais difícil descrever do que ver! Por exemplo, tente descrever o procedimento de calçar o seu sapato.
- Muitos usuários não têm a liberdade de admitir que eles não seguem os procedimentos prescritos; então, o que eles dizem para você pode ou não ser o que eles realmente fazem.

- Cada usuário possui seu próprio padrão de trabalho profundamente impregnado, aplicam "quebra-galhos" ou caminhos próprios de implementação, os quais mascaram os problemas reais para quem os observa.
- É impossível para qualquer desenvolvedor antecipar todas as questões que devem ser perguntadas ou para quaisquer usuários saberem quais questões que os desenvolvedores deveriam perguntar.

Para atender a essas causas em particular, o simples ato de "interpretar papéis" (role playing) pode ser extremamente efetivo. É barato e normalmente muito rápido. Normalmente, em uma hora ou meio dia a mágica terá sido realizada.

Como Interpretar o Papel

Na forma mais simples de interpretar papéis, o desenvolvedor, o analista, e, potencialmente, todos os membros da equipe de desenvolvimento simplesmente tomam o lugar do usuário e executam as atividades de trabalho do cliente. Por exemplo, no caso do problema de entrada dos pedidos de venda da Habilidade de Equipe 1, alertamos para o fato de que os pedidos de venda imprecisos era o principal condutor para o alto custo de desperdícios e, através disso, criar vantagens do problema. Quando nós olhamos para o processo de pedidos de venda, nós esperamos encontrar uma série de passos diferentes e fontes de erros. Existem ao menos duas maneiras de descobrir as raízes causas.

1. Use a técnica da espinha de peixe que descrevemos, junto com a entrevista de usuários, e analise os pedidos de venda que reconhecidamente possuem erros. Quantifique os erros por tipos e ataque os mais graves no projeto de um novo sistema. Isso pode fornecer um entendimento quantitativo para o problema e provavelmente será bastante efetivo.

   No entanto, isso não lhe dá a impressão qualitativa do problema, aquela que possa, talvez, mudar tanto a percepção quanto a sua estratégia de solução. A fim de conseguir isso, talvez exista uma maneira mais eficiente para verdadeiramente entender o problema.

2. O desenvolvedor/analista pode experimentar os problemas e imprecisões intrínsecas ao sistema de entrada de pedidos existente, apenas *sentando-se e entrando com alguns pedidos de venda*. A experiência obtida em 1 hora irá mudar, para sempre, o entendimento da equipe sobre o problema.

Nós podemos dizer que as experiências obtidas no role playing, podem ficar com os desenvolvedores pelo o resto da vida. Nossa visão pessoal do mundo foi mudando de acordo com tais experiências, incluindo os simples papéis de "soldar uma peça complexa da maneira que se supõem que os robôs façam", "misturar compostos farmacêuticos num depurador de fluxo laminar", "identificar uma televisão comercial com apenas quatro resumos de tela e uma trilha de áudio", "usar ferramentas de software para gerenciamento de requisitos imaturos", entre muitos outros. Nós aprendemos sempre alguma coisa, e desenvolvemos uma grande empatia com os usuários com os quais estivemos!

Experimente!

Técnicas Similares ao Role Playing

Naturalmente, o role playing não funcionam em todas as situações. Em muitos casos, o papel do usuário é mínimo, e o problema a ser resolvido é algoritmicamente, ao invés de funcionalmente, intenso. E, em muitos casos, simplesmente não é prática. Nós não gostaríamos de ser o primeiro voluntário para fazer o papel de "paciente" num cenário de cirurgia auxiliada por equipamentos eletrônicos ou de "operador de uma fábrica nuclear no período noturno" ou "piloto de um 747". Em muitos casos, outras técnicas nos aproximam das experiências de usuários sem ter que "sangrar" pelos lados.

Roteiro de Acompanhamento

Um roteiro de acompanhamento é um "jogo" no papel.

Num roteiro de acompanhamento (*scripted walkthrough*), cada participante segue um roteiro que define um papel específico dentro do "jogo". O acompanhamento irá demonstrar qualquer mal-entendido nos papéis, falta de informação disponível sobre um ator ou subsistema, ou falta de um comportamento específico necessário para que atores possam ter sucesso em seus esforços.

Por exemplo, certa vez nós construímos um roteiro de acompanhamento que mostrava como o professor e estudantes deviam interagir com um dispositivo automático de prova-pontuação usado na sala de aula. Nós usamos um protótipo do dispositivo e tínhamos membros da equipe e representantes do cliente interpretando os papéis de alunos e de professor. O roteiro de acompanhamento continha muitas cenas, tais como "pontuação dos estudantes em suas provas" e "professor pontuando um grande lote de provas durante a aula". O roteiro de acompanhamento foi muito útil para perceber o ambiente de uma sala de aula, e a equipe aprendeu algumas coisas novas durante a experiência. Foi muito divertido.

Uma das vantagens do roteiro de acompanhamento é que o roteiro pode ser modificado e reexecutado quantas vezes forem necessárias até que os atores façam direito. O roteiro pode também ser reusado para educar novos membros da equipe. Ele pode ser modificado e reusado quando o comportamento do sistema for alterado. Assim, o roteiro se transforma num storyboard vivo do projeto.

Cartões CRC (Classe-Responsabilidade-Colaboração)

Um derivado do role playing é freqüentemente aplicado como parte de um esforço de análise orientado a objetos. Neste caso especial de role playing, é dado a cada participante um conjunto de cartões descrevendo a classe, ou objetos; as responsabilidades, ou comportamentos; e colaborações, ou com quem os objetos se comunicam, de cada entidade sendo modelada. Essas colaborações envolvem entidades do domínio do problema, tais como usuários, botões, lâmpadas e elevador de carros, ou objetos que vivem no domínio da solução, tais como Botão Indicador de Luz, Windows MDI e Elevador de Carro.

Quando o ator inicia um dado comportamento, todos os participantes seguem o comportamento definido em seus cartões. Quando o processo falhar devido a uma falta de informação ou quando uma entidade precisar falar com uma outra e a colaboração não estiver definida, os cartões são modificados e o jogo recomeça.

Por exemplo, no caso de estudos HOLIS, existirá um momento em que a equipe precisará entender as interações entre os três subsistemas a fim de determinar como o sistema irá cooperar para atingir os objetivos e entender quais serão os requisitos derivados criados. Uma forma de fazer isso é utilizar os cartões CRC como roteiro de acompanhamento. Um membro da equipe deve fazer o papel de um subsistema, ou ator, e então a equipe deve acompanhar através do use case, ou cenário. Aqui está como um use case pode ser exercitado:

John (Chave de Controle):	O proprietário acabou de pressionar um botão que controla um banco de luz. Ele ainda está pressionando a chave. Eu enviarei ao Bob uma mensagem assim que a chave estiver totalmente pressionada, e enviarei ao Bob uma mensagem a cada segundo em que a chave ficar pressionada.
Bob (Unidade de Controle Central):	Quando eu receber a primeira mensagem, mudarei o estado da saída de Desligado para Ligado. Quando eu receber a segunda mensagem, sei que o proprietário está reduzindo a intensidade do banco de luz. Assim, a cada mensagem recebida, reduzirei a intensidade da luz em 10%. John, não esqueça de me dizer qual botão está pressionado.
Mike (Lâmpada):	Eu estou fisicamente conectado à saída do dimmer. Eu percebo o dimmer como nós falamos.

Nota: Na elucidação das necessidades do usuário, o processo CRC é direcionado para os comportamentos externos que são aparentes aos atores, mas esta técnica pode também ser usado para projetar sistemas orientados a objetos. Neste exercício, o foco estava em entender o trabalho interno do software, não nas interações com o ambiente externo. No entanto, até nesses casos, a técnica ajuda a descobrir erros ou enganos de requisitos do sistema.

Você deve ter notado um efeito colateral interessante. Os jogadores invariavelmente acham que existem falhas ou deficiências no roteiro e os corrigem, resultando normalmente no aperfeiçoamento no entendimento do sistema.

Sumário

O role playing é uma excelente técnica, embora não a tenhamos visto em uso com muita freqüência. Por que? As razões são muitas. Existe o fator desconforto. Esta técnica não nos motiva ao ter que fazer mal feito um simples pedido de venda enquanto nossos clientes ou pessoas que entram com um pedido de venda nos observam. Além disso, existe o fator delicado e confuso: Ser forçado a interagir com pessoas reais ao invés de com um teclado, nos faz sair da nossa zona de conforto – afinal de contas, nós queremos é compilar classes teóricas, deixem que nossos colegas participem de um drama!

No entanto, não há dúvidas de que, se nós conseguirmos vencer essa pequena dificuldade, o role playing será uma das técnicas mais acessíveis e eficazes para assistir no descobrimento de requisitos.

Capítulo 15

Prototipação

Pontos chaves

- A prototipação é especialmente efetiva para atacar as síndromes "Sim, mas" e "Ruínas Desconhecidas".
- Um protótipo de requisitos de software é uma implementação parcial do sistema de software, construído para auxiliar os desenvolvedores, usuários e clientes a melhor entenderem os requisitos do sistema.
- Crie protótipos para requisitos "confusos": aqueles que, embora conhecidos ou implícitos, suas definições são pobres e mal compreendidas.

Os protótipos de software, como encarnações de um sistema de software, demonstram uma porção da funcionalidade de um novo sistema. Dado o que discutimos até aqui, nós esperamos que fique obvio de que a prototipação pode ser muito útil para descobrir necessidades do usuário. Usuários podem tocar, sentir e interagir com o protótipo do sistema de maneira que nenhuma das outras técnicas pode fornecer. De fato, a prototipação pode ser extremamente efetivo para atacar tanto a síndrome "Sim, mas" ("Isso não é exatamente o que eu queria"), quanto a síndrome das "Ruínas Desconhecidas" ("Agora que eu vi, eu tenho um outro requisito a adicionar").

Tipos de Protótipos

Protótipos podem ser categorizados de várias maneiras. Por exemplo, Davis (1995a) categoriza os protótipos como descartável versus evolucionário versus operacional, vertical versus horizontal, interface de usuário versus algorítmico, entre outras. O tipo de protótipo que você escolhe depende do problema que você está tentando resolver através da construção do protótipo.

Por exemplo, se o risco do seu projeto é baseado principalmente na viabilidade da abordagem tecnológica – isso, simplesmente nunca foi feito antes e você não está certo se a tecnologia aplicada pode atingir as metas de desempenho ou de rendimento – você pode querer desenvolver um *protótipo arquitetural* que principalmente demonstre a viabilidade da tecnologia a ser usada. Um protótipo arquitetural pode ainda ser *descartável* versus *evolucionário*. "Descartável" implica que o propósito do esforço é apenas para provar a viabilidade; assim, você poderá usar qualquer atalho, técnicas alternativas, simulações, ou qualquer outra coisa para atingir esse propósito. Quando você tiver terminado, você simplesmente o descarta, mantendo apenas o conhecimento apreendido nesse exercício. "Evolucionário" implica que você implementou o protótipo na mesma arquitetura que você pretende usar no sistema final, e que você será capaz de construir o sistema final a partir da evolução do protótipo.

Se a principal área de risco de seu projeto é a interface do usuário, por contrate, você irá querer desenvolver um *protótipo de requisitos*, usando qualquer tecnologia que permita a você, desenvolver interfaces do usuário muito rapidamente. A Figura 15–1 ilustra uma árvore de decisão que você pode usar para selecionar um tipo de protótipo que faça mais sentido para o seu projeto.

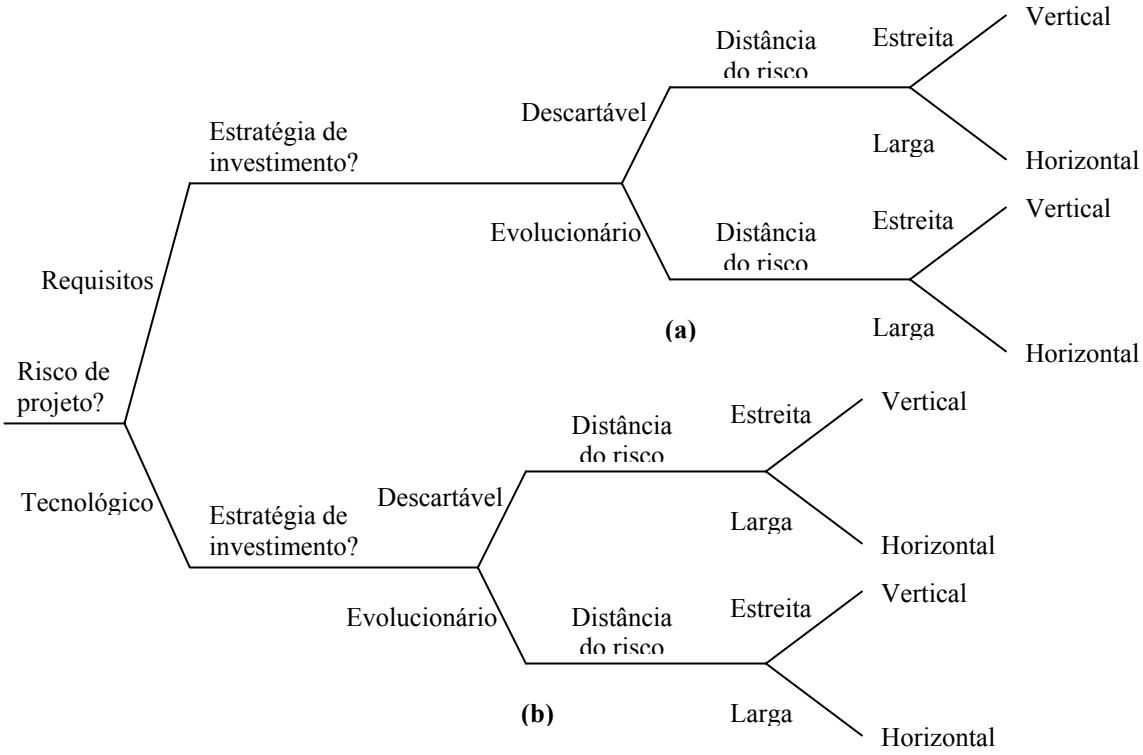


Figura 15–1 Árvore de decisão para seleção do tipo de protótipo: (a) protótipos de requisitos; (b) protótipos arquiteturais

Protótipos de Requisitos

Para propósitos de elucidação de requisitos, nós nos concentramos sobre os tipos de protótipos sob o ramo superior desta árvore. Definimos um protótipo de requisitos de software como:

> *uma implementação de um sistema de software, construída para ajudar desenvolvedores, usuários e clientes a melhor entender os requisitos do sistema.*

Com o propósito de elucidar requisitos, nós freqüentemente escolhemos construir um protótipo "descartável, horizontal, interface do usuário". "Horizontal" implica que nós iremos tentar construir uma grande quantidade de funcionalidade do sistema; um protótipo vertical, por outro lado, constroem-se apenas alguns requisitos de maneira qualitativa. "Interface de usuário" implica que nós iremos construir principalmente a interface do sistema para seus usuários ao invés de implementar a lógica e os algoritmos que residem dentro do software ou

prototipar interfaces para outros dispositivos ou sistemas. Como uma ferramenta de elucidação, os protótipos:

- Construídos por desenvolvedores, podem ser usados para obter confirmação de que o desenvolvedor entendeu os requisitos.
- Construídos por desenvolvedores, podem ser usados como um catalisador para encorajar o cliente a pensar em mais outros requisitos.
- Construídos pelo cliente, podem comunicar requisitos ao desenvolvedor.

Em todos os três casos, a meta é construir o protótipo de maneira que consuma poucos recursos. Se ficar muito caro construir, pode ser melhor construir o sistema real!

Muitos protótipos de software tendem a ser protótipos de requisitos e são usados principalmente para capturar aspectos da interface do usuário do sistema a ser construído. Existem provavelmente duas razões para isso:

1. A emergência de uma pletora[4] de ferramentas baratas e amplamente disponíveis para construir interfaces de usuários rapidamente.
2. Para sistemas cujas interfaces de usuário sejam intensas, um protótipo de interface de usuário revela, também, muitos outros requisitos, tais como as funções que são fornecidas ao usuário, quando cada função está disponível aos usuários e quais funções não são acessíveis aos usuários.

No entanto, precisamos estar certos de que a disponibilidade de ferramentas não nos leve a prototipar partes do sistema que não apresentem, inicialmente, riscos muito altos.

O que Prototipar

Como vamos saber qual porção do sistema devemos prototipar? Numa situação típica, nosso entendimento das necessidades dos usuários irá variar desde o bem conhecido e fácil de verbalizar até o totalmente desconhecido (Figura 15–2).

Figura 15–2 O sistema de estoque inicial, com atores identificados

Os requisitos bem-conhecidos podem ser óbvios no contexto do domínio da aplicação e a experiência do usuário e da equipe com sistemas desse tipo. Por exemplo, se estivermos simplesmente estendendo um sistema existente, teremos claramente a idéia de como a maioria das necessidades por novas funcionalidades deve ser. Os requisitos bem-conhecidos e bem-entendidos não precisam ser prototipados, a menos que sejam necessários para ajudar a visualizar o contexto de outras necessidades de usuários; construí-los irá consumir os já parcos recursos

[4] Superabundância qualquer, que produz efeito nocivo.

existentes, e como já estão bem-compreendidos, aprenderemos muito pouco com eles.

Os requisitos desconhecidos, no entanto, são as "Ruínas Desconhecidas" que nós desejamos conhecer. Infelizmente, nós não podemos realmente prototipá-los, se pudéssemos, não seria desconhecido! Assim, sobra como alvo de prototipação as partes "Confusas" que estão no meio. Esses requisitos podem ser conhecidos ou implícitos, mas sua definição e entendimento são pobres.

Construindo o Protótipo

A escolha da tecnologia usada para construção do protótipo depende das decisões tomadas considerando a árvore de decisão da Figura 15–1. Por exemplo, a escolha por um protótipo descartável da GUI nos leva a escolher qualquer tecnologia que seja barata e rápida para implementar exemplos de GUIs.

Se um protótipo evolucionário for selecionado, você deve escolher a linguagem e o ambiente de desenvolvimento que será utilizado para produzir a implementação final. Você também terá que fazer esforços significativos para projetar a arquitetura de software do sistema, bem como aplicar quaisquer padrões de codificação ou processos de software que você usará para criar o sistema. Caso contrário você terá que evoluir um sistema que fundamentalmente falha em um ou mais desses aspetos. Neste caso, você pode ter criado um protótipo descartável por acidente! Ou, pior, a qualidade do sistema implantado estará, para sempre, comprometida pelo seu protótipo de requisitos bem-intencionado.

Avaliando os Resultados

Depois que o protótipo estiver construído, ele deve ser exercitado pelos usuários num ambiente que simule, tanto quanto possível, o ambiente de produção no qual o sistema final será usado. Dessa forma, ambientes e outros fatores externos que afetam os requisitos do sistema também se tornarão óbvios. Além disso, é importante que existam vários tipos de usuários exercitem o protótipo, caso contrário os resultados serão preconceituosos.

Os resultados do processo de prototipação dividem-se em duas partes:

1. Necessidades confusas tornam-se melhor entendidas.
2. Ao exercitar o protótipo, inevitavelmente elucida respostas "Sim, mas" do usuário; assim, necessidades anteriormente desconhecidas tornam-se conhecidas. Simplesmente por enxergarem um conjunto de comportamentos, os usuários passam a entender outros requisitos que devem ser impostos ao sistema.

De qualquer forma, a prototipação virtualmente sempre produz resultados. Assim, você deve normalmente prototipar qualquer aplicação nova ou inovadora. O truque é assegurar que o retorno obtido no conhecimento dos requisitos faça valer o investimento realizado. Isso porque queremos, freqüentemente, prototipar – ou ao menos implementar nossos primeiros protótipos – rapidamente, utilizando técnicas baratas e disponíveis. Ao limitar o investimento, nós maximizamos o retorno sobre o investimento de obter o entendimento dos requisitos.

Sumário

Devido aos protótipos de software demonstrarem uma parte da funcionalidade desejada de um novo sistema, eles podem ser ferramentas efetivas para ajudar a refinar os requisitos reais do sistema. Eles são efetivos porque usuários podem *interagir* com um protótipo em seu ambiente, o qual é tão próximo ao mundo real quanto se puder chegar sem desenvolver o software de produção.

Você deve selecionar sua técnica de prototipação com base na probabilidade de um tipo de risco estar presente em seu sistema. Supõe-se que os protótipos de requisitos sejam baratos e fáceis de desenvolver, e que eles ajudem a eliminar grande parte dos riscos de requisitos de seu projeto.

Entre as várias possibilidades de investimento, você deve investir somente o necessário em seu protótipo. O uso de várias técnicas de prototipação, ou melhor, o uso combinado de diversas de técnicas de prototipação, tem se mostrado extremamente efetivo em auxiliar a equipe de projeto a desenvolver um melhor entendimento das reais necessidades de um sistema de software.

Sumário da Habilidade de Equipe 2

Três "síndromes" contribuem com os desafios de entender as reais necessidades dos usuários e de outros stakeholders. A síndrome do "Sim, mas", das "Ruínas Desconhecidas" e do "Usuário e o Desenvolvedor" são metáforas que nos ajudam a entender melhor os desafios que estão à nossa frente e fornecem um contexto para as técnicas de elucidação que desenvolvemos para entender as necessidades dos usuários.

Porém, como são raros os casos onde a equipe recebe especificações efetivas de requisitos do sistema que eles terão que construir, eles têm que sair e obter as informações que precisam para garantir o sucesso. O termo "elucidação de requisitos" descreve este processo, onde a equipe deve assumir um papel mais ativo.

Para ajudar a equipe nessa missão, uma variedade de técnicas pode ser usada para atacar problemas e melhorar o entendimento das reais necessidades dos usuários e de outros stakeholders:

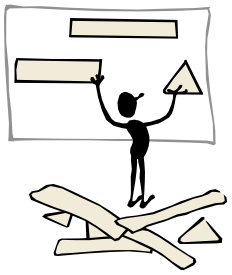
- Entrevistas e questionários
- Workshop de requisitos
- Brainstorming e redução de idéias
- Storyboarding
- Use cases
- Role playing
- Prototipação

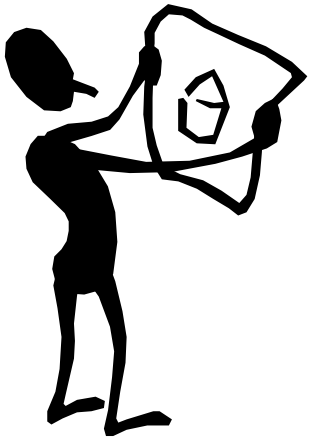
Embora nenhuma das técnicas seja perfeita para todas as circunstâncias, cada um representa um meio pró-ativo de impulsionar o entendimento das necessidades do usuário e converter requisitos "confusos" para requisitos que sejam "melhor conhecidos". Embora todas estas técnicas funcionem em certas circunstâncias, o nosso favorito é a técnica do workshop/brainstorming.

Habilidade de Equipe 3

Definindo o Sistema

- **Capítulo 16**: Organizando as Informações de Requisitos
- **Capítulo 17**: O Documento da Visão
- **Capítulo 18**: O Campeão





Na Habilidade de Equipe 1, nós desenvolvemos as habilidades que fazem com que a equipe concentre-se na análise do problema. Ao fazer isso, nós conseguimos compreender totalmente o problema a ser resolvido antes que fossem investidos quaisquer esforços sérios na solução. Nós estivemos completamente concentrados no domínio do problema.

Na Habilidade de Equipe 2, descrevemos um conjunto de técnicas que a equipe pode usar para entender as necessidades do usuário. Essas necessidades do usuário vivem no topo da nossa pirâmide de requisitos, indicando que são essas as informações que devemos entender inicialmente, por serem as mais críticas e dirigirem tudo o que vier após esse entendimento.

A quantidade de informações que devemos gerenciar aumenta rapidamente conforme nos movemos para baixo da pirâmide.

Na Habilidade de Equipe 3, partimos do espaço do problema para chegar ao espaço da solução, sendo que o nosso foco se concentrará na *definição do sistema* que será construído para atender as necessidades dos stakeholders. Conforme nos movemos para baixo da pirâmide, (veja Figura 1), a quantidade de informações aumenta. Por exemplo, podem ser necessárias grandes quantidades de características do sistema para atender a uma única necessidade do usuário. Começamos, também, a fornecer especificidades adicionais para facilitar a definição do comportamento do sistema; desse modo, a quantidade de informações que devemos gerenciar aumenta.

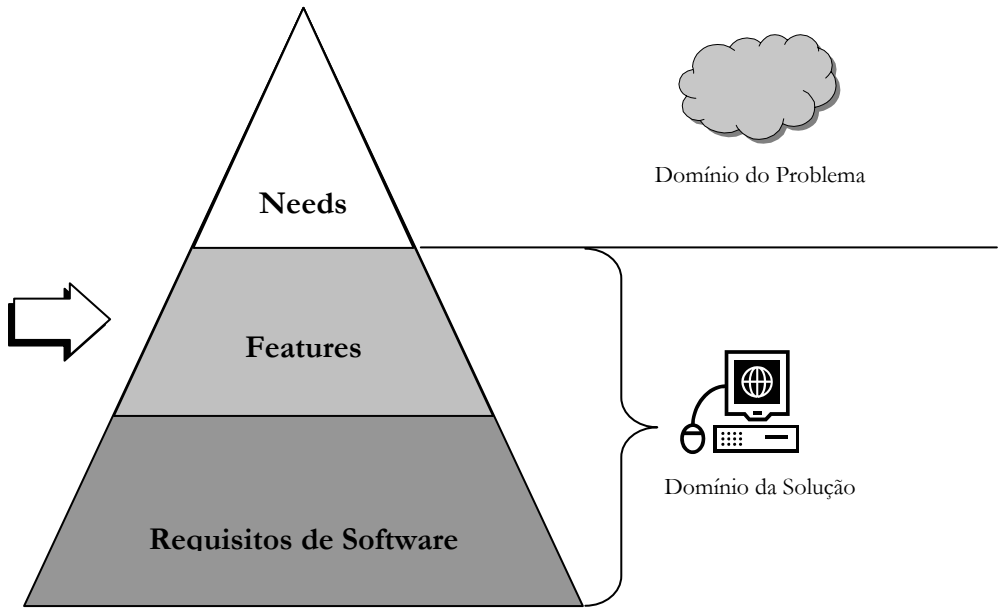


Figura 1 Características na pirâmide de requisitos

Além disso, a equipe deve, agora, também se preocupar com vários outros assuntos que são únicos no espaço da solução, mas que têm um pouco a ver com o domínio do problema. Por exemplo, se nós estivermos desenvolvendo um produto de software para ser vendido ao usuário, devemos nos preocupar com empacotamento, instalação e licenciamento, cada um dos quais podem ser únicos para a solução que estamos fornecendo. Se estivermos desenvolvendo um sistema para atender as necessidades de IS/IT na empresa do cliente, pode ser que precisemos nos preocupar com os requisitos de implantação e manutenção, os quais são de pouco ou nenhum interesse ao usuário que não está usando o sistema.

No entanto, devemos ainda manter a abstração suficientemente alta, para não nos perdermos em detalhes muito rapidamente, não nos permitindo ver a "floresta entre as árvores". Além disso, é importante parar por um segundo, para organizar as informações de requisitos, antes de nos movermos de seção da pirâmide de requisitos de software. Isso será visto na Habilidade de Equipe 5, Refinando a Definição do Sistema. Por agora, iremos cobrir a organização das informações de requisitos (Capitulo 16), a definição de uma visão (Capítulo 17) e a organização de *nossa equipe* para atacar os desafios de gerenciamento de requisitos para o sistema (Capítulo 18).

Capítulo 16

Organizando Informações de Requisitos

Pontos chaves

- Para aplicações não triviais, requisitos devem ser capturados e registrados numa base de dados de documentos, modelos e ferramentas.
- Tipos diferentes de projetos requerem diferentes técnicas de organização de requisitos.
- Sistemas complexos exigem especificação de requisitos para cada subsistema.

Requisitos devem ser capturados e documentados. Se você for um desenvolvedor solitário de um sistema no qual também será o usuário e o mantenedor, você pode pensar em fazer o projeto e a codificação imediatamente após identificar suas necessidades. No entanto, poucos desenvolvimentos de sistemas são assim tão simples. É muito provável que desenvolvedores e usuários sejam mutuamente exclusivos, e que stakeholders, usuários, desenvolvedores, analistas, testers, arquitetos e outros membros da equipe estejam envolvidos. Todos devem chegar a um acordo sobre o sistema que será desenvolvido.

A realidade do orçamento e do cronograma não torna razoável satisfazer todas as necessidades do usuário em qualquer particular release. Inevitavelmente, problemas inerentes à comunicação, devido ao esforço envolvendo múltiplas pessoas, irão exigir que um documento escrito seja produzido para que todas as partes possam concordar e consultar.

Documentos que definem o produto a ser construído são normalmente chamados de especificação de requisitos. A *especificação de requisitos* de um sistema ou aplicação descreve o comportamento externo desse sistema.

Nota: Por simplicidade e para refletir a convenção histórica, usamos o termo *documento* de forma genérica nesta seção, mas requisitos podem estar presentes num documento, numa base de dados, num modelo use-case, repositório de requisitos ou numa combinação desses elementos. Como veremos nos próximos capítulos, um "pacote de requisitos de software" pode ser usado para conter esta informação.

Mas, os requisitos raramente podem ser definidos num único documento monolítico por uma série de razões:

- O sistema pode ser muito complexo.

- As necessidades dos clientes são documentadas antes da documentação detalhada dos requisitos.
- O sistema pode ser um membro de uma família de produtos relacionados.
- O sistema a ser construído satisfaz a apenas um subconjunto de todos os requisitos identificados.
- É necessário que as metas de marketing e de negócio estejam separadas dos detalhes de requisitos do produto.

Em qualquer um dos casos, você precisará manter múltiplos documentos e, naturalmente, considerar vários casos especiais:

- Um documento "pai" define os requisitos do sistema como um todo, incluindo hardware, software, pessoas e procedimentos; e um outro define os requisitos apenas para a peça de software. Normalmente, o primeiro documento é chamado de *especificação dos requisitos do sistema*, enquanto que o segundo é chamado de *especificação dos requisitos de software*, ou SRS para abreviar.
- Um documento define as características (features) do sistema em termos gerais, e o outro define os requisitos em termos mais específicos. Com freqüência, o primeiro documento é chamado *documento da visão*, o segundo é chamado de *especificação dos requisitos de software*.
- Um documento define o conjunto total de requisitos para uma família de produtos, e o outro define os requisitos de apenas uma aplicação específica e para um release específico. O primeiro documento é normalmente chamado de *documento dos requisitos da família de produtos*, ou *documento da Visão da família de produtos*, já, o segundo, é chamado de *especificação de requisitos de software*.
- Um documento descreve os requisitos gerais de negócio e ambiente de negócio no qual o produto irá residir, e o outro define o comportamento externo do sistema a ser construído. Normalmente, o primeiro documento é chamado de *documento dos requisitos de negócio*, ou *documento dos requisitos de marketing*, já, o segundo, é chamado de *especificação dos requisitos de software*.

As seguintes seções descrevem o que fazer em cada um dos casos. Alguns ou todos estes casos podem ser combinados; por exemplo, um documento pode conter o conjunto total de requisitos, a partir do qual subconjuntos selecionados podem ser usados para gerar releases específicos, bem como todos os requisitos de negócio.

Organizando Requisitos de Sistemas Complexos de Hardware e Software

Embora este volume tenha como principal foco os requisitos de software, é importante reconhecer que eles são apenas um subconjunto do processo de gerenciamento de requisitos na maioria dos esforços de desenvolvimento de sistemas. Como descrevemos no Capítulo 6, alguns sistemas são tão complexos que a única forma de visualizá-los e construí-los é como um sistema de subsistemas, os quais por sua vez são visualizados como sistemas de subsistemas, e assim por diante, como ilustra a Figura 16–1. Num caso extremo, tal como uma

aeronave de transportes, o sistema pode ser composto de centenas de subsistemas, que por sua vez possuem componentes de hardware e software.

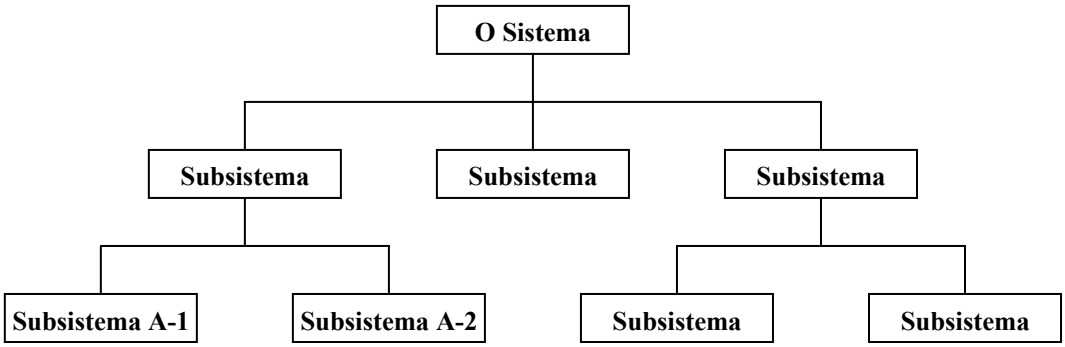


Figura 16–1 Um sistema de sistemas

Nesses casos, é criada uma especificação de requisitos ao nível sistêmico que descreve o comportamento externo do sistema, tais como capacidade de combustível, velocidade de decolagem ou altitude máxima, sem conhecer ou referenciar qualquer de seus subsistemas. Como descrevemos no Capítulo 6, uma vez que se tenha chegado a um acordo sobre os requisitos do sistema, uma atividade de engenharia de sistemas é executada. A *engenharia de sistemas* refina um sistema em subsistemas, descrevendo as interfaces detalhadas entre subsistemas, e alocando cada um dos requisitos do nível sistêmico para um ou mais subsistemas. A arquitetura do sistema resultante descreve esse particionamento e interfaces entre sistemas.

Depois, uma especificação de requisitos é desenvolvida para cada subsistema. Estas especificações devem descrever, por completo, o comportamento externo do subsistema, sem referenciar quaisquer de seus subsistemas. Este processo provoca o surgimento de uma nova classe de requisitos, os requisitos derivados. Este tipo de requisito nem de longo descreve o comportamento externo do *sistema*, ao invés disso descreve o *comportamento externo* do novo subsistema. Assim, o processo de projeto de sistemas cria novos requisitos para os subsistemas dos quais o sistema é composto. Em particular, as interfaces entre esses subsistemas tornam-se requisitos chaves: essencialmente, um contrato entre um subsistema e os outros, ou uma *promessa* de desempenho conforme acordado.

Uma vez que se tenha chegado a um acordo sobre os requisitos, o projeto do sistema é novamente executado se necessário, dividindo cada subsistema em subsistemas e desenvolver especificações de requisitos para cada um. O resultado é uma hierarquia de especificações, como ilustra a Figura 16–2.

Em todos os níveis, requisitos do nível anterior são alocados para as especificações apropriadas do nível posterior. Por exemplo, o requisito da capacidade de combustível é alocado para o subsistema de controle de combustível e para o subsistema de armazenamento de combustível, e novos requisitos são descobertos e definidos quando apropriado.

Como ilustra a Figura 16–3, especificações que por sua vez são refinadas em especificações adicionais de subsistema são chamadas de especificações dos requisitos de sistemas, ou *especificações dos requisitos ao nível sistêmico*. As

especificações do último nível, isto é, aqueles que não serão futuramente decompostas, normalmente correspondem a subsistemas de software-somente ou de hardware-somente e são chamados especificações dos requisitos de software ou especificações dos requisitos de hardware, respectivamente. Além disso, toda especificação dos requisitos da Figura 16–3 pode precisar experimentar um processo evolucionário conforme os detalhes forem mais bem compreendidos.

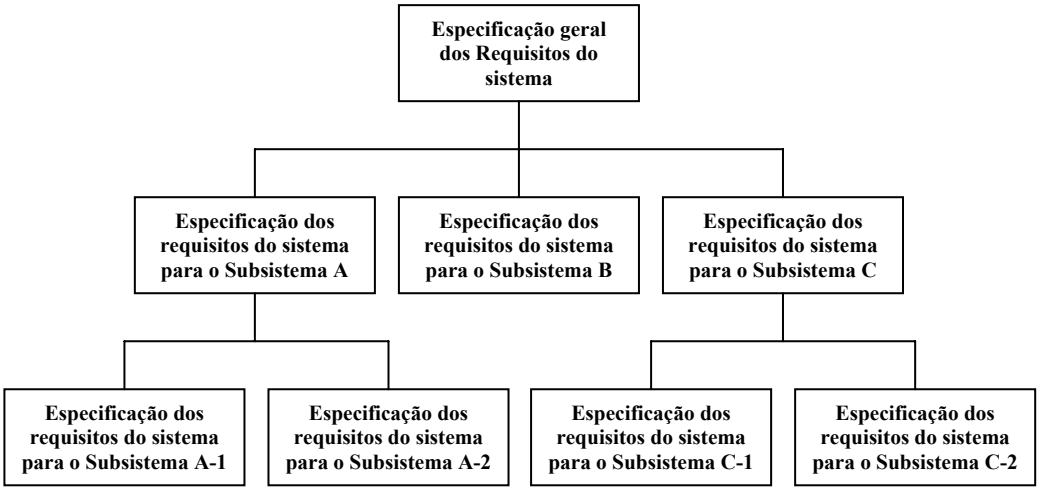

Especificação geral dos Requisitos do sistema
Especificação dos requisitos do sistema para o Subsistema A
Especificação dos requisitos do sistema para o Subsistema B
Especificação dos requisitos do sistema para o Subsistema C
Especificação dos requisitos do sistema para o Subsistema A-1
Especificação dos requisitos do sistema para o Subsistema A-2
Especificação dos requisitos do sistema para o Subsistema C-1
Especificação dos requisitos do sistema para o Subsistema C-2

Figura 16–2 Hierarquia de especificações resultante do projeto de sistemas

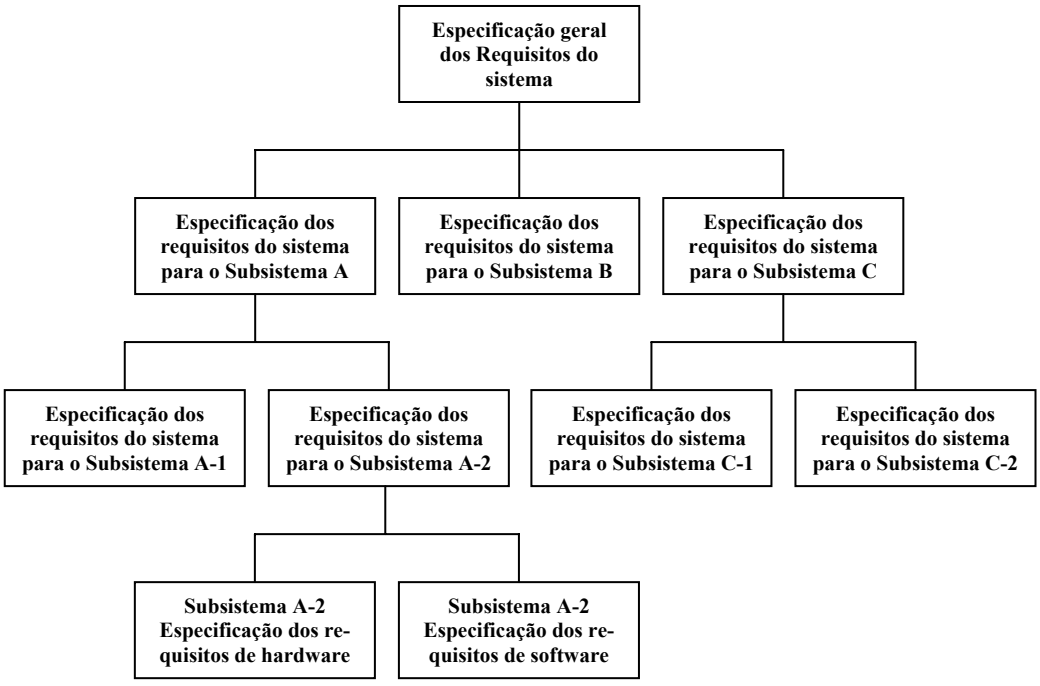

Especificação geral dos Requisitos do sistema
Especificação dos requisitos do sistema para o Subsistema A
Especificação dos requisitos do sistema para o Subsistema B
Especificação dos requisitos do sistema para o Subsistema C
Especificação dos requisitos do sistema para o Subsistema A-1
Especificação dos requisitos do sistema para o Subsistema A-2
Especificação dos requisitos do sistema para o Subsistema C-1
Especificação dos requisitos do sistema para o Subsistema C-2
Subsistema A-2 Especificação dos requisitos de hardware
Subsistema A-2 Especificação dos requisitos de software

Figura 16–3 Hierarquia de especificações resultante do projeto de sistemas, incluindo os níveis de software e hardware

Requisitos de Organização para Família de Produtos

Muitas indústrias constroem conjuntos de produtos intimamente relacionados que contém funcionalidades em comuns, mas com cada um contendo algumas características próprias. Exemplos de tas famílias de produtos pode ser: sistemas de controle de inventário, máquinas de respostas telefônicas, sistemas de alarme contra ladrões, entre outros.

Como exemplo, suponha que você esteja construindo um conjunto de produtos de software, cada um dos quais compartilham algumas funcionalidades, mas que podem precisar compartilhar dados ou de algum jeito se comunicar com um outro quando em uso. Em tais casos, você pode considerar a seguinte abordagem:

- Desenvolver um *documento da Visão* da família de produtos que descreva a maneira como os produtos pretendem trabalhar em conjunto e outras características que podem ser compartilhadas.
- Para entender melhor o modelo de uso compartilhado, você pode também desenvolver um conjunto de *use cases* mostrando como os usuários irão se interagir com as várias aplicações executando em conjunto.
- Desenvolver uma *especificação dos requisitos de software comuns* que defina os requisitos específicos para funcionalidades compartilhadas, tais como estruturas de menu e protocolos de comunicação.
- Para cada produto da família, desenvolver um *documento da Visão*, a *especificação dos requisitos de software*, e um *modelo use-case* que defina as funcionalidades específicas.

A organização resultante é mostrada na Figura 16–4.

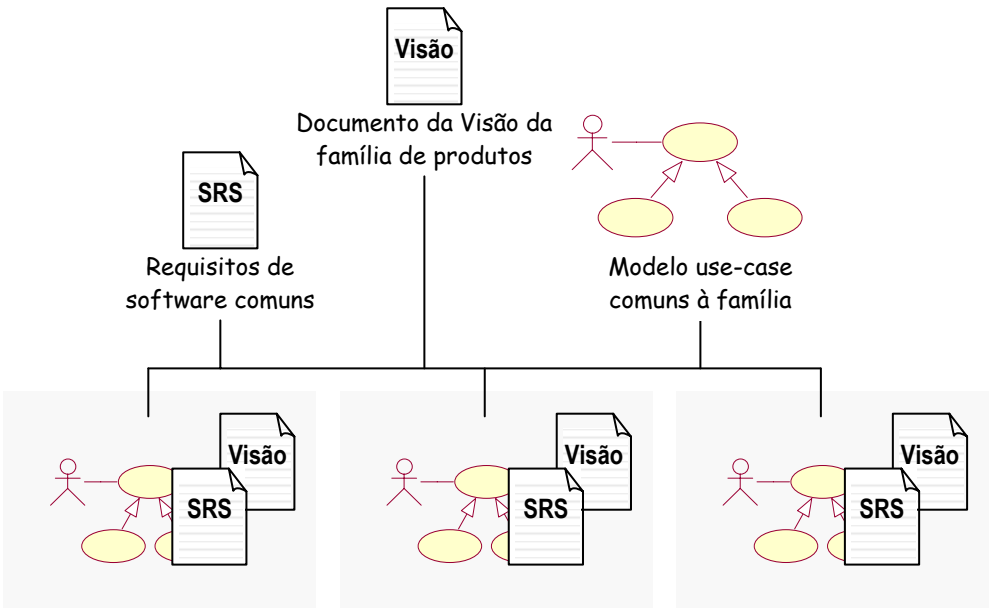


Figura 16–4 Organização dos requisitos para uma família de produtos de software

As especificações dos requisitos para cada membro individual pode conter referências – *links ou* "*traced from* " – para o documento da família de produtos ou pode reproduzir todos os requisitos a partir desse documento. A vantagem da primeira abordagem é que as mudanças nos requisitos pertencentes a todos os membros da família podem ser realizadas em apenas um lugar. No segundo caso, você poderá precisar usar uma ferramenta de requisitos para gerenciar essas dependências, senão, será necessário examinar manualmente cada membro específico do documento de requisitos toda vez que o documento pai for alterado.

Sobre os Requisitos "Futuros"

Poucos esforços de desenvolvimento têm o luxo de ter um conjunto de requisitos estáveis ou estar apto a construir um sistema que satisfaça todos os requisitos conhecidos. Durante qualquer processo de elucidação de requisitos, surgem requisitos que não são apropriados para a construção do próximo release.

Pode não ser apropriado incluir tais requisitos numa especificação de requisitos; não podemos nos permitir fazer qualquer confusão sobre quais requisitos serão e quais não serão implementados. Por outro lado, não é apropriado descartá-los, porque eles representam produtos de um trabalho de valor adicionado[5], e queremos colher tais requisitos para futuros releases. E, o mais importante, os projetistas de sistemas podem muito bem projetar o sistema de maneira diferente sabendo que certos tipos requisitos futuros têm grandes chances de serem considerados no próximo release. A melhor coisa é registrar ambos os tipos de requisitos em algum lugar do documento, mas deixando claramente identificados aqueles requisitos que estão planejados para o atual release.

Requisitos de Negócio e de Marketing versus Requisitos de Produto

O planejamento de um novo produto não ocorre num mundo técnico desprovido de considerações de negócio. Devem ser feitas considerações sobre oportunidades de mercado, mercado alvo, empacotamento do produto, canais de distribuição, funcionalidades, custo de marketing, disponibilidade de recursos, margens, amortização sobre um grande número de cópias vendidas, entre outras.

Tais considerações devem ser documentadas, mas elas não pertencem às especificações de requisitos. Algumas organizações usam um documento dos requisitos de mercado (DRM) para facilitar a comunicação entre o pessoal de gerenciamento, marketing e desenvolvedores, e para auxiliar nas decisões de negócio sobre inteligência de mercado, incluindo todas as decisões importantes de "vamos ou não-vamos em frente". O DRM também fornece aos clientes e desenvolvedores uma rápida verificação da comunicação, para adiantar o entendimento do produto em seus termos mais gerais, e estabelecer o escopo geral do produto. O DRM deve responder as seguintes questões:

[5] A diferença entre o valor da mercadoria produzida por um assalariado e o salário que lhe é pago; lucro em longo prazo que não é visto de imediato.

- Quem é o cliente?
- Quem são os usuários?
- Qual é o mercado no qual pretendemos vender?
- Como o mercado está segmentado?
- Os requisitos do usuário estão nesses diferentes segmentos?
- Quais classes de usuários existem?
- Quais necessidades o produto satisfaz?
- De que tipo é o produto?
- Quais são os principais benefícios do produto; por que alguém deve comprá-lo?
- Quem são os concorrentes?
- O que diferencia o produto de nossos concorrentes?
- Em que ambiente o sistema será usado?
- Qual será o custo de desenvolvimento?
- Com que preço você pretende vender o produto?
- Como o produto será instalado, distribuído e mantido?

O Caso de Estudos

No Capítulo 6, nós executamos algumas atividades da engenharia de sistemas no sistema HOLIS, o nosso sistema de automação de iluminação residencial. Neste ponto, não sabemos, ainda, muito sobre o HOLIS, mas provavelmente sabemos o suficiente para fazer uma primeira tentativa em organizar as informações de nossos requisitos.

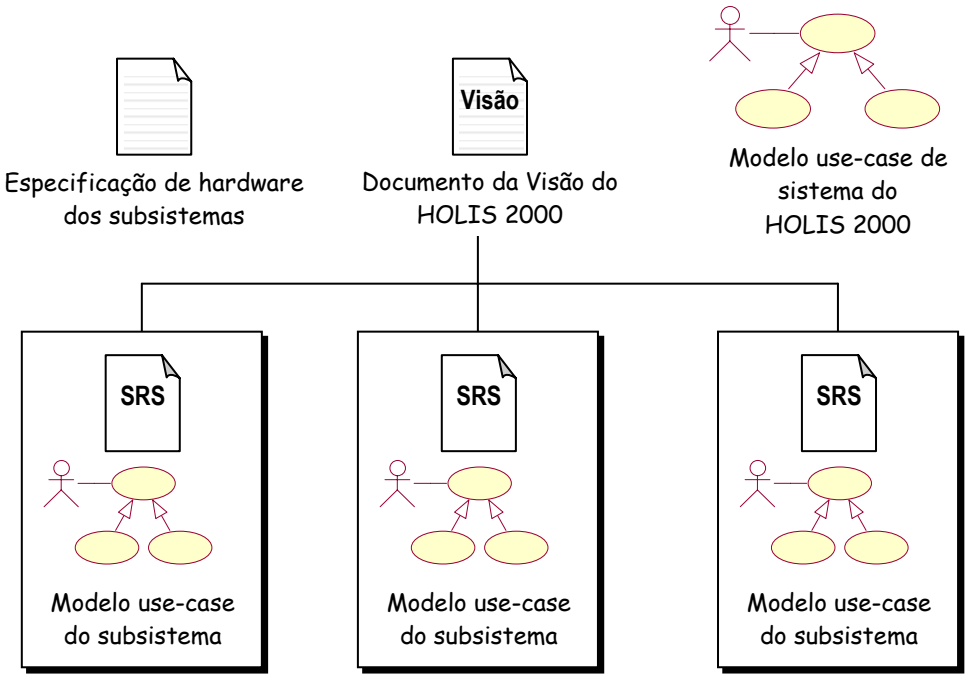


Figura 16–5 Organização das informações dos requisitos do HOLIS

A Figura 16–5 ilustra que a equipe está usando os seguintes elementos para descrever os requisitos para o HOLIS:

- O *documento da Visão* que irá conter visões de curto e longo prazo do HOLIS, incluindo requisitos e características básicas do sistema que estão sendo propostos.
- O *modelo use-case de sistema* registra os use cases através dos quais os vários atores do sistema interagem com o HOLIS.
- Após alguns debates, a equipe decidiu documentar os requisitos de hardware – tamanho, peso, potência, empacotamento – para os três subsistemas do HOLIS numa única *especificação dos requisitos de hardware*.
- Como cada subsistema do HOLIS é intensa em software, a equipe decidiu desenvolver uma *especificação de requisitos de software* para cada um dos três subsistemas, bem como um *modelo use-case* de cada subsistema para mostrar como cada subsistema interage com os vários atores.

Você terá a oportunidade de ver esses artefatos de requisitos desenvolvidos posteriormente conforme avançamos no caso de estudos nos próximos capítulos. Um exemplo de cada um foi incluído no Apêndice A.

Sumário

Neste capítulo, nós examinamos uma variedade de documentos de requisitos para sistemas de diferentes complexidades. No entanto, em muitos casos, o gerenciamento de requisitos eventualmente concentra-se sobre um único subsistema de software, produto de software de prateleira, ou aplicações standalone. Exemplos desses produtos podem ser o Microsoft Excel, Rational ClearCase, produtos de fornecedores ISV, ou o sistema de controle de iluminação HOLIS.

Nos próximos capítulos, nós iremos fazer um “zoom” sobre o processo de definição de requisitos para uma única aplicação de software tal que possamos demonstrar mais claramente como o processo de gerenciamento de requisitos funciona.

Capítulo 17

O Documento da Visão

Pontos chaves

- O documento da Visão descreve a aplicação em termos gerais, incluindo descrições do mercado alvo, dos usuários do sistema, e das características da aplicação.
- O documento da Visão define, num alto nível de abstração, tanto o problema quanto a solução.
- Virtualmente, todos os projetos de software irão se beneficiar ao ter um documento da Visão.
- O documento da Visão Delta concentra-se no que foi mudado.

Este capítulo concentra-se no documento da Visão. Como nosso colega Philippe Kruchten disse recentemente, "Se me fosse permitido desenvolver apenas um único documento, modelo, ou algum outro artefato para sustentar um projeto de software, a minha escolha, em resumo, seria o documento da Visão muito bem confeccionado".

Documento da Visão para o meu projeto

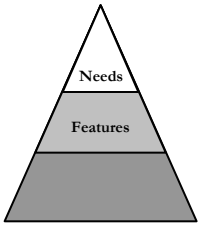
O documento da Visão combina, dentro de um único documento, alguns elementos modestos tanto do documento dos requisitos de mercado quanto do documento dos requisitos de produto. Nós queremos desenvolver este documento em particular pelas seguintes razões:

1. Todo projeto precisa de um documento da Visão.
2. O documento da Visão nos ajudará a demonstrar o processo de requisitos, bem como alguns elementos chaves desse processo registrados nesse documento.

O documento da Visão descreve a aplicação em termos gerais, incluindo descrições do mercado alvo, dos usuários do sistema e das características da aplicação. Por anos, vimos constatando a utilidade deste documento, tanto que o desenvolvimento deste documento tornou-se, para nós, um padrão de melhor prática na definição de uma aplicação de software.

Componentes do Documento da Visão

O documento da Visão, talvez *o único* documento mais importante de um projeto de software, captura as necessidades do usuário, as características do sistema, e outros requisitos comuns do projeto. Como tal, o escopo do documento da Visão estende sobre os dois primeiros níveis da pirâmide de requisitos, através dos quais define-se, num nível alto de abstração, tanto o *problema* quando a *solução*.



Escopo do documento da visão

Para um produto de software, o documento da Visão também serve de base para a discussão e contrato entre as três principais comunidades de stakeholders do projeto:

1. O departamento de marketing, que serve como representante de clientes e usuários, e que no final será o responsável pelo sucesso do produto após a sua liberação.
2. A equipe de projeto que desenvolverá a aplicação.
3. A equipe de gerenciamento, a qual será responsável pelos resultados do esforço de negócio.

O documento da Visão é poderoso porque ele representa a estrutura (gestalt[6]) do produto a partir de todas as perspectivas significativas de forma breve, abstrata, legível e manejável. Como tal, o documento da Visão é o principal foco nas fases iniciais do projeto, e muito investimento é feito no processo de obter as informações que irão satisfazer o conhecido retorno nas fases posteriores.

Uma vez que, virtualmente, todos os projetos de softwares podem se beneficiar do documento da Visão, nós o descreveremos em detalhes. Embora o nosso exemplo esteja orientado para um produto específico de software, não será difícil modificá-lo para o seu contexto particular de produto.

A Figura 17–1 fornece um exemplo do documento da Visão com um breve comentário. Estes comentários foram usados, com personalizações, em centenas de produtos de software e numa grande variedade de aplicações de software. Uma versão totalmente comentada deste documento consta no Apêndice B.

Em resumo, o documento da Visão é uma descrição concisa de todas as coisas que você considera ser mais importante sobre o produto ou aplicação. O documento da Visão deve ser escrito, com um nível de detalhes e de claridade suficientes, para permitir que seja prontamente lido e compreendido pelos stakeholders do projeto.

1. Introdução

Esta seção deve fornecer um resumo completo do documento da Visão.

1.1. Propósito do Documento da Visão

Este documento reúne, analisa e define as necessidades do usuário e características do produto em alto nível.

1.2. Visão Geral do Produto

Estabelecer o propósito da aplicação, sua versão, e as novas características a serem liberadas.

1.3. Referências

Fornecer uma lista completa de todos os documentos referenciados neste documento.

Continua na próxima página

Figura 17–1 Modelo de documento da Visão para um produto de software

[6] Corrente na psicologia

2. Descrição do Usuário

Descrever brevemente a perspectiva dos usuários do seu sistema.

2.1. Demografia do Usuário/Mercado

Resumir os principais mercados demográficos que motivaram suas decisões de produto.

2.2. Perfil do Usuário

Descrever brevemente os potenciais usuários do sistema.

2.3. Ambiente do Usuário

Detalhar o ambiente de trabalho dos usuários alvo.

2.4. Principais Necessidades do Usuário

Listar os principais problemas ou necessidades que são percebidos pelo usuário.

2.5. Alternativas e Concorrentes

Identificar quaisquer alternativas que o usuário percebam que estejam disponíveis.

3. Visão Geral do Produto

Fornecer uma visão de alto nível das capacidades do produto, interfaces com outras aplicações e configurações do sistema.

3.1. Perspectiva do Produto

Fornecer um diagrama de blocos do produto ou sistema e suas interfaces com o ambiente externo.

3.2. Declaração da Posição do Produto

Fornecer uma declaração geral sumarizando, em alto nível, a posição única que o produto pretende preencher no mercado. Moore (1991) recomenda que siga o seguinte formato:

Para	(cliente alvo)
Que	(declaração da necessidade ou oportunidade)
O (nome do produto)	é um (categoria do produto)
Que	(declaração dos benefícios principais, isto é, razões para convencer a comprar)
Diferente	(principais alternativas da concorrência)
Nosso produto	(declaração das principais diferenças)

3.3. Resumo das Capacidades

Resumir os maiores benefícios e características que o produto fornece.

Benefícios do Cliente	Características Suportadas
Benefício 1	Característica
Benefício 2	Característica
Benefício 3	Característica

Figura 17–1 Modelo de documento da Visão para um produto de software (continua)

3.4. Suposição e Dependências

Listar as suposições que, se alteradas, irão afetar a visão do produto.

3.5. Custo e Preço

Registrar quaisquer restrições de custo e preços que sejam relevantes.

4. Atributos de Características

Descreva as características que serão usadas para avaliar, rastrear, priorizar e gerenciar as características. A seguir estão algumas sugestões:

Estado	Proposto, Aprovado, Incorporado
Prioridade	Resultado do voto acumulado; ordem de prioridade, ou Crítico, Importante, Útil
Esforço	Baixo, Médio, Alto; equipe-semana; ou pessoa-mês
Risco	Baixo, Médio, Alto
Estabilidade	Baixo, Médio, Alto
Release alvo	Número da versão
Associado a	Nome
Razão	Campo texto

5. Características do Produto

Esta seção do documento lista as características do produto.

5.1. Característica #1

5.2. Característica #2

6. Principais Use Cases

Descreve alguns use cases principais, talvez aqueles que são arquiteturalmente significantes ou aqueles que irão mais prontamente ajudar o leitor a entender como o sistema pretende ser usado.

7. Outros Requisitos de Produtos

7.1. Padrões Aplicáveis

Lista todos os padrões que o produto deve cumprir.

7.2. Requisitos do Sistema

Definir quaisquer requisitos necessários para sustentar a aplicação.

7.3. Licenciamento e Instalação

Descreva quaisquer requisitos de instalação que também afete a codificação ou que crie a necessidade para separar o software de instalação.

7.4. Requisitos de Desempenho

Use esta seção para detalhar os requisitos de desempenho.

8. Requisitos de Documentação

Descreva as documentações que devem ser desenvolvidas para sustentar a implantação da aplicação com sucesso.

Figura 17–1 Modelo de documento da Visão para um produto de software (continua)

8.1. Manual do Usuário

Descreva o propósito e o conteúdo do manual do usuário do produto.

8.2. Help Online

Requisitos do help online, dicas de ferramentas, entre outros.

8.3. Guia de Instalação, Configuração e Arquivos Leia-me

8.4. Endereçamento e Empacotamento

8.5. Glossário

Figura 17–1 Modelo de documento da Visão para um produto de software

O Documento da "Visão Delta"

O desenvolvimento e gerenciamento do documento da Visão podem representar papel chave para o sucesso ou fracasso de um projeto de software, pois fornece um lugar comum para as atividades de stakeholders, clientes, usuários, gerentes de produto e gerentes de marketing. Freqüentemente, até o gerente executivo da empresa estará envolvido no seu desenvolvimento e revisão. Manter o documento da Visão compreensível e gerenciável é uma importante habilidade de equipe, pois irá gerar grandes benefícios à produtividade do projeto como um todo.

Para auxiliar nesse processo, é útil manter o documento da Visão tão curto e conciso quanto possível. Isso não é difícil, principalmente nas primeiras releases do documento, quando todos os itens do documento são novidades para o projeto ou quando devem ser reiniciados num novo contexto da aplicação.

No entanto, nos futuros releases, você pode descobrir que é contraproducente repetir características e outras informações que não tenham sido alteradas num contexto particular do projeto, tais como perfil do usuário e mercado-alvo, e já tenham sido incorporadas nos releases anteriores.

Documento da Visão para o Release 1.0

No caso de um novo produto ou aplicação, provavelmente todos os elementos do documento da Visão devem ser desenvolvidos e elaborados. Itens que estiverem fora do contexto do projeto podem ser removidos do documento e você não terá que preenchê-lo. O documento da Visão deve conter ao menos os seguintes itens (veja a Figura 17–2):

- Informações gerais e introdutórias
- Uma descrição dos usuários do sistema, mercado-alvo, e características pretendidas na versão 1.0
- Outros requisitos, tais como de regulação e ambiental
- Características futuras que tenham sido elucidadas, mas que não serão incorporadas no release 1.0

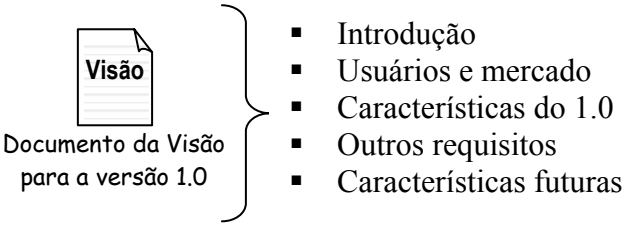


Figura 17–2 Documento da Visão v1.0

Este documento serve de base para o release 1.0 e dirige os requisitos e use cases mais detalhados do sistema, os quais serão muito mais elaborados.

Documento da Visão para a Versão 2.0

Com a evolução do projeto, características vão sendo mais bem definidas; normalmente, isso significa que elas serão mais bem elaboradas no documento da Visão. Além disso, novas características serão descobertas e adicionadas ao documento. Assim, o documento tende a crescer, tornando-se mais valoroso para a equipe. Quando chegarmos a abordar a versão 2.0, certamente iremos querer manter esse documento que tão bem nos serviu. O próximo passo lógico na evolução do projeto e deste documento é "extrair" as características futuras que foram incluídas na versão 1.0 do documento e não implementadas, e programá-las para a versão 2.0. Em outras palavras, queremos encontrar e "promover" algumas características que serão importantes para o release 2.0. Você pode, também, querer programar um outro workshop de requisitos ou outros processos de elucidação a fim de descobrir novas características que entrarão na programação do release 2.0 e algumas novas características futuras que precisarão ser registradas e documentadas. Algumas dessas características serão óbvias, com base no retorno do cliente (feedback), outras irão ser definidas com base na experiência da equipe. Em qualquer dos casos, registrar essas novas características descobertas na versão 2.0 do documento da Visão, ou como programadas para ser incluídas na versão 2.0 ou como novas características futuras.

Provavelmente você descobrirá que algumas características implementadas na versão 1.0 não liberaram o valor pretendido, seja devido a mudanças ambientais ocorridas durante o processo, quando tais características deixaram de ser necessárias, seja pela substituição por novas características, ou talvez pelo cliente que simplesmente não necessitou dessas características como ele pensava que iria necessitar. Em qualquer caso, você provavelmente descobrirá que será necessário *remover* algumas características no próximo release. Como registrar esses "anti-requisitos"? Simplesmente utilize o documento da Visão para registrar o fato de que uma particular característica deve ser removida no próximo release.

Conforme a equipe desenvolve seu trabalho através do processo, ela descobre que o documento cresce com o tempo. Isso é tão natural quanto definir um sistema que está em crescimento. Infelizmente, você pode descobrir que, com o tempo, o documento torna-se mais difícil de se ler e entender. Por que? Porque o as informações do documento são mais antigas e contêm muitas informações que não mudaram desde o último release. Por exemplo, as declarações do

posicionamento do produto e usuários-alvo provavelmente não mudaram, assim como as características 25-50 implementadas na versão 1.0 que vivem no documento da Visão da versão 2.0.

Assim, nós sugerimos o conceito de *documento da Visão Delta*. O *documento da Visão Delta* concentra-se somente em duas coisas: *o que mudou e quaisquer outras informações que devam ser incluídas para definir o contexto*. Esta última informação é incluída ou como um lembrete para a equipe da visão do projeto ou porque novos membros da equipe necessitam do contexto para o seu entendimento.

O resultado é um documento da Visão Delta que agora *tem como foco principal descrever o que é novo e descrever o que mudou* no release. O foco, apenas nas coisas que mudaram, é uma técnica de aprendizagem que é extremamente benéfica ao tratar com sistemas de informações complexas. Este modelo é ilustrado na Figura 17–3.

- A versão 1.0 é o nosso *ponto de partida para o entendimento*; é o que nos conta tudo que precisamos para conhecer o nosso projeto.
- A versão 2.0 define o que é diferente nesse release.
- Tomados em conjunto, a visão 1.0 e a visão delta 2.0 definem a "*definição completa do produto*".

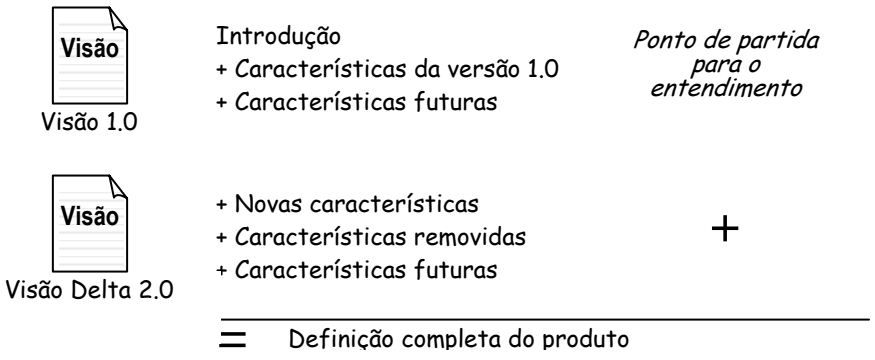


Figura 17–3 O Documento da Visão Delta

A duas versões devem ser utilizadas em conjunto sempre que for necessária a definição completa do produto, tanto para requisitos de regulação quanto de clientes, por exemplo, e é óbvio que isso é importante para os novos membros da equipe. No entanto, quando características forem removidas, você irá ler características que estão na versão 1.0, mas que não aparecem na versão 2.0. Assim, se necessário, siga essa trilha de auditoria sempre que você precisar ressuscitar a definição completa.

Se e quando isso se tornar complicado, é fácil reunir o conteúdo da versão 1.0 e o Delta da versão 2.0 num novo documento da Visão 2.0 que represente um quadro do projeto compreensivo e completo.

Naturalmente, não precisamos ficar restritos sobre esta definição ou ao que cada documento contém. Em várias circunstâncias, verificamos que é conveniente usar a Visão Delta apenas para pequenas mudanças – tais como para a versão 1.1 ou 1.2 – e iniciar com uma declaração "completa do produto", rigorosamente

revisada a cada grande release – tais como nas versões 2.0 ou 3.0. Nesses casos, a aplicação do documento da Visão Delta deve ajudar a melhor gerenciar o processo de requisitos por permitir que sua equipe concentre-se o que "realmente interessa" em cada momento específico.

O Documento da Visão Delta num Ambiente de Sistema Legado

Raramente, se não nunca, é prático documentar completamente os requisitos de um sistema legado de larga-escala.

Um dos problemas mais melindrosos do gerenciamento de requisitos está em aplicar a habilidade de gerenciamento de requisitos para evoluir sistemas legados do tipo IS/IT. Raramente, se não nunca, existem especificações de requisitos completa ou adequada para as milhões de linhas de código de centenas de pessoas por anos de investimento refletidos nesses sistemas. Não é prático parar e redocumentar o passado. Por vezes, quando você faz isso, a necessidade passa, e você falha em sua missão por escrever requisitos históricos, quando você deveria estar escrevendo o código!

Assim, se você está começando do zero ou com uma documentação mínima, você deve prosseguir com base no melhor esforço, usando qualquer recurso que você possa encontrar ao seu redor – código, especificações, membros da equipe com um conhecimento histórico – para chegar a um entendimento do que o sistema faz agora. Nossa recomendação nesses casos é aplicar o processo da Visão Delta e definir suas características e use cases ao redor das *mudanças* que você estiver fazendo no sistema legado. Ao seguir esse processo, você e sua equipe podem se concentrar naquilo que é novo e naquilo que é diferente no próximo release, e seu cliente e sua equipe irão obter os benefícios de um processo de requisitos bem-gerenciado. Além disso, o registro dos requisitos que você cria irá fornecer um caminho de documentação para outros que se seguirem.

Capítulo 18

O Campeão

Pontos chaves

- O campeão produto mantém a visão do projeto.
- Todo projeto necessita de um indivíduo campeão ou de uma pequena equipe campeã para defender o produto.
- Nos ambientes do produto de software, o campeão do produto irá normalmente vir do marketing.

No Capítulo 1, nós analisamos os desafios de projetos e descobrimos uma variedade de causas raízes, com o gerenciamento de requisitos estabelecido próximo ao topo da lista. No Capítulo 17, nós definimos o documento da Visão como um documento embrionário dentro de um ciclo-de-vida de software complicado; este documento ataca diretamente os desafios dos requisitos e é um dos documentos que você pode olhar, em qualquer momento, para ver o que o produto, aplicação ou sistema faz ou não faz. Enfim, o documento da Visão representa a essência do produto e deve ser defendido como se o sucesso do projeto dependesse disso.

Em algum ponto, a questão com razão se transforma, "Mas quem desenvolve e mantém esse documento tão importante? Quem gerencia as expectativas do cliente? Quem negocia com a equipe de desenvolvimento, o cliente, o departamento de marketing, o gerente de projeto, e com os executivos da empresa que têm demonstrado tamanho interesse, entusiasmado com o projeto agora que se aproxima o prazo final?".

Em virtualmente todos os projetos de sucesso que estivemos envolvidos – desde o projeto de veículos de aventura que colocam borboletas na barriga de todos os visitantes até projetos de ventiladores de suporte-de-vida que sustentam dez vidas sem uma única falha de software – tinha um campeão. Nós podemos olhar para o passado desses projetos e apontar um indivíduo, e nos casos de grandes projetos, uma pequena equipe de pessoas, os quais interpretam um papel "maior que as suas próprias vidas". O campeão mantém a visão do produto (ou sistema ou aplicação) na frente de suas mentes como se ela fosse a única coisa mais importante de suas vidas. Eles comem, dormem e sonham sobre a visão do projeto.

O Papel do Campeão do Produto

O campeão do produto pode ter uma grande variedade de títulos: gerente de produtos, gerente de projeto, gerente de marketing, gerente de engenharia, gerente de tecnologia da informação, líder de projeto. Mas independente do título, o trabalho é o mesmo. É um grande trabalho. O campeão deve:

- Gerenciar o processo de elucidação e ficar tranqüilo quando são descobertos requisitos suficientes.
- Gerenciar as entradas conflitantes de todos os stakeholders.
- Fazer as concessões necessárias para encontrar o conjunto de características que liberem o mais alto valor para o maior número de stakeholders.
- Apropriar-se da visão do produto.
- Defender o produto.
- Negociar com gerentes, usuários e desenvolvedores.
- Defender frente a requisitos estranhos.
- Manter uma "saudável excitação" entre o que o cliente deseja e o que o a equipe de desenvolvimento pode liberar dentro do prazo do release.
- Ser o representante oficial entre o cliente e a equipe de desenvolvimento.
- Gerenciar expectativas dos clientes, gerentes executivos e os departamentos internos de marketing e engenharia.
- Comunicar as características do release para todos os stakeholders.
- Revisar as especificações de software para assegurar que eles conformam com a visão representada pelas características.
- Gerenciar as prioridades das mudanças e a adição e remoção das características.

Esta pessoa é a única a quem o documento da Visão pode ser realmente confiada, e encontrar o campeão correto para esse propósito é a chave de sucesso ou falha do projeto.

O Campeão do Produto num Ambiente de Produto de Software

Certa vez nós conduzimos um workshop para um provedor de serviços online confrontando requisitos desafiadores. Quando nós chegamos na parte do tutorial enfatizando a noção de campeão do produto, a sala ficou inquieta, e o clima mudou nitidamente. Nós perguntamos para as 25 pessoas presentes, incluindo os desenvolvedores, engenheiros seniores e gerentes de marketing, como essas duras decisões eram realizadas em seus ambientes. Depois de alguns momentos, ficou claro que ninguém fazia essas decisões. Após discussões entre eles, o melhor que o grupo pôde descrever foi um "grupo cego", seguindo informações que vem e vão como as marés. Ninguém tinha a responsabilidade de tomar decisões. Ninguém decidia quem seria suficientemente bom.

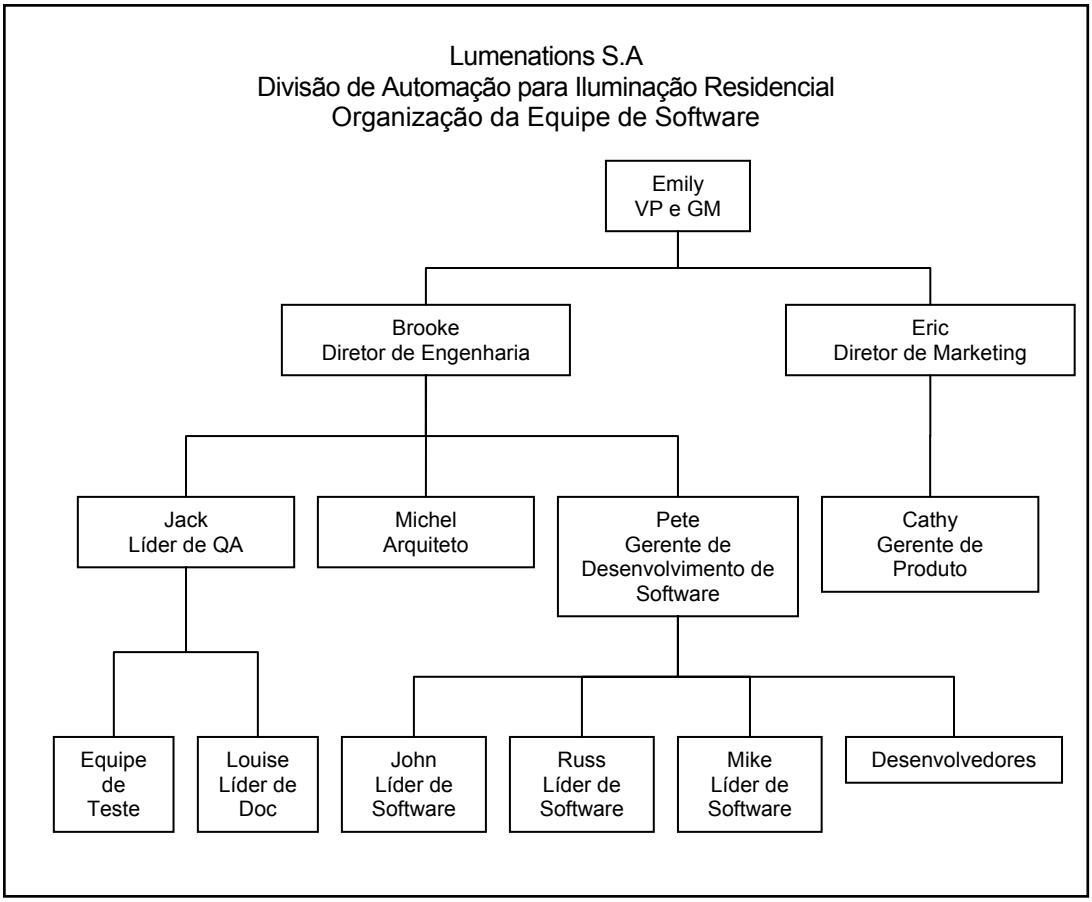
Eventualmente, a equipe voltava-se para trás, para um executivo de marketing, por respostas, talvez porque esse indivíduo tinha mais informações no processo. Ele olhou ao redor por um momento e então disse: "*Você sabe o que mais me apavora nesta equipe? Eu posso pedir que algumas novas características sejam adicionadas em algum momento do processo, e ninguém nunca me dirá não. Como esperar que consigamos liberar, em algum momento, um produto?"*.

Ficou claro, após esta explanação, de que o cliente não sabia quando um produto seria liberado. É verdade também que a história demonstra que houve uma "inabilidade" extremamente dolorosa deste cliente. A companhia desenvolveu-se de uma cultura tradicional de IT e tinha-se transformado num provedor de

serviços on-line recentemente. A noção de uma aplicação como um *produto de software* era novo. Os membros da equipe não tinham precedentes para guiá-los através do processo.

Embora não exista uma forma correta de organizar e determinar um campeão do produto, talvez nós possamos olhar para o nosso caso de estudos para obter alguma sugestão. Afinal de contas, nós o modelamos de acordo com uma equipe de projeto viva e efetiva.

Na Figura 18–1, é a Cathy, a *gerente de produto*; quem faz o papel de campeão do produto. Note que, neste caso, o gerente de produto se reporta diretamente com o marketing ao invés de se reportar para a organização de engenharia. Isso é muito normal em empresas de produtos de software, ao menos em tese, que gerentes de produto estejam mais próximos aos clientes, pois são eles que determinarão o sucesso ou a falha do projeto.



Talvez o gerente de produto se reporte à gerência de marketing porque, no final de contas, é ele que tem a responsabilidade pelo "Número", isto é, pela receita associada com cada produto liberado. É assim que deve ser; o marketing é, em última análise, o responsável pelas vendas e deve assim, ser responsável pelo mercado e pelas decisões difíceis; caso contrário suas prestações de contas não terão sentido.

Como você classificaria os papéis e responsabilidades nessa mescla de requisitos, tecnologias, clientes e desenvolvedores? Imagine o seguinte diálogo, uma variação do que ocorreu recentemente numa nova empresa que foi classificar seus papéis:

Gerente de Produto (o Campeão):	Deverá possuir as características A, B e C, e você deve construir usando a tecnologia X.
Gerente de Desenvolvimento:	Eu acho que deveria ter a característica D e não a C, e nos usar a tecnologia Y.
Gerente de Produto:	Eu disse que tem que ter A, B e C. Afinal de contas, sou eu que tenho que atender às quotas de venda, e é o que preciso para atender às necessidades dos clientes. Se você quiser se responsabilizar, você pode adicionar a característica D, desde que você ainda atenda o cronograma.
Gerente de Desenvolvimento:	Hummm (*pensando melhor no significado de gastar seu tempo ajudando sua equipe a atender à quota*). Eu não tenha certeza se isso é uma boa idéia. Nós implementaremos A, B e C. Mas você se responsabiliza se, na prática, isso não funcionar?
Gerente de Produto:	(antevendo o conhecimento de como escrever o código nas próximas 48 horas ao invés de preparar o lançamento para o mercado) Hummm, não, eu acho que isso não é uma boa idéia. Você pode construir utilizando a tecnologia que achar mais apropriada.
Gerente de Desenvolvimento:	Concordo. Você decide as características e nós escolhemos a tecnologia; essa é a melhor combinação de nossas habilidades e responsabilidades.
Gerente de Produto:	Está combinado.

Esse diálogo não tem nada de mais; apenas explana o senso comum, mas é incrível o quão freqüente é encontrar empresas que não têm claro esta responsabilidade, como no caso da empresa provedora de serviços on-line.

De alguma forma, o ambiente de fornecimento de software independente (ISV) é simples: O cliente é externo, e nós normalmente temos uma organização de marketing razoavelmente sofisticada a qual usamos para elucidar os requisitos e determinar quem é o responsável pelo equilíbrio de todas as necessidades conflitantes. Um cliente cujas necessidades não são atendidas, simplesmente não será um cliente. Embora isso não seja uma boa coisa, ao menos eles não vão perder tempo blasfemando.

Este não é o caso em empresas de IS/IT. Não existe departamento de marketing, todos os seus clientes trabalham com você, e certamente se tornarão preguiçosos depois que constatar que o release satisfaz seus desejos.

Onde encontrar nosso campeão em tal ambiente? Talvez nós possamos novamente aprender com um exemplo. Numa empresa, um novo sistema de apoio empresarial foi desenvolvido para prover acesso global, 24 horas do dia, aos clientes registrados para suporte a vendas e gerenciamento de licenças. O exercício de análise do problema identificou os seguintes stakeholders: marketing corporativo, televendas, licenciamento e suporte, marketing de unidade de negócio, gerência financeira da unidade de negócio, execução de pedidos, e garantia. Cada um destes departamentos tinha fortes necessidades, mesmo assim, estava claro que nem todas as necessidades poderiam ser atendidas. A questão "Quem é o dono do documento da Visão?", apresenta-se como uma metáfora para a questão "Quem gostaria de fazer uma grande mudança em sua carreira limitada tentando gerenciar este projeto?".

Na análise, ficou claro que nenhuma das lideranças da equipe de desenvolvimento tinha autoridade para fazer tais decisões, mas ainda é necessário um campeão. Neste caso, a equipe decidiu chamar Tracy, a líder de projeto atual, como a campeã do produto, dando-lhe autoridade para produzir e organizar os requisitos. Ela se apropriou do documento da Visão. Ela entrevistou os usuários, estabeleceu suas prioridades relativas, e coletou dados num formato orientado-a-característica. Mas um comitê diretor especial, ou comissão de controle de mudanças (CCM) de projeto, fora também imediatamente estabelecida para o projeto. A CCM foi constituída de três executivos seniores, cada um com a responsabilidade numa área funcional.

Inicialmente, Tracy facilitou o processo de tomada de decisão da CCM estabelecendo prioridades relativas para as releases iniciais. Desde então, a CCM, e somente a CCM, tem a autoridade de adicionar ou remover características, com recomendações realizadas pela campeã do produto. Desta maneira, existe apenas uma campeã, Tracy, e os resultados da elucidação e a visão do projeto existem em sua cabeça e no Documento da Visão, mas a responsabilidade de tomar as decisões difíceis foi delegada à CCM. A campeã tinha apenas que ver que as características acordadas tinham sido apropriadamente elaboradas e comunicadas à equipe de desenvolvimento.

Desde que Tracy foi delegada para dirigir o processo em conjunto com a CCM, a qual incluía membros da gerência sênior, obtendo respaldo necessário para tomar decisões, o projeto alcançou seu sucesso e foi usado como um modelo organizacional para novos projetos. Alocou-se um novo campeão para cada novo projeto. Isso criava a oportunidade para que as pessoas pudessem crescer e se desenvolverem individualmente. O papel de campeão do produto tornou-se muito importante dentro da empresa. Não podemos nos esquecer, naturalmente, da CCM. Para cada novo projeto, estabelecia-se uma nova CCM, com base nos temas de cada nova release e na organização que poderia ser mais afetada diretamente.

Embora não existam prescrições que possam ser seguidas para criar um campeão de projeto, é extremamente importante para a sua equipe identificar um, promovê-lo, ou delegar para alguém que já esteja, aparentemente, liderando o processo. Então, é responsabilidade da equipe assisti-lo de todas as maneiras possíveis no gerenciamento de requisitos da aplicação. Isso irá ajudar a atingir o sucesso. Além disso, se você não ajudar essa pessoa a atingir o sucesso, ela poderá convidá-la para que você seja o campeão do projeto num próximo projeto.

Sumário da Habilidade de Equipe 3

Na Habilidade de Equipe 3, nós nos movemos do entendimento das necessidades do usuário para iniciar a definição da solução. Ao fazermos isso, nós demos a engatinhada inicial para sairmos do domínio do problema, a ilha do usuário, e entrarmos no domínio da solução, o lugar onde nós trabalhamos para definir um sistema que solucione o problema que temos em mãos.

Nós também aprendemos que sistemas complexos necessitam de estratégias claras para gerenciar informações de requisitos, e algumas maneiras de organizar informações de requisitos. Nós reconhecemos que nós realmente temos uma hierarquia de informações, começando pelas necessidades dos usuários, passando pelas características, chegando finalmente aos requisitos de software mais detalhados representado pelos use cases ou formas tradicionais de expressões. Notamos também que a hierarquia reflete o nível de abstração com o qual nós enxergamos o espaço do problema e o espaço da solução.

Nós, então, ampliamos a nossa visão do processo de definição utilizando como exemplo uma aplicação de software standalone e investimos algum tempo na definição de seu Documento da Visão. É crucial que todos os projetos mantenham o Documento da Visão, modificando-o para atender às particularidades de uma aplicação de software da companhia.

Nós reconhecemos que sem um campeão – alguém que defenda os requisitos da aplicação e atenda às necessidades dos clientes e da equipe de desenvolvimento – não temos como garantir que decisões difíceis sejam tomadas. Decisões difíceis provavelmente serão as de abandonar, postergar ou relaxar requisitos por pressões de cronograma. Assim, decidimos indicar um, sacramentar alguém: alguém que ficasse responsável pelo Documento da Visão e pelas características que ele contém. Por sua vez, o campeão e a equipe irão delegar as decisões difíceis para uma comissão de controle de mudanças, assegurando que as mudanças de requisitos sejam pensadas antes de serem aceitas.

Tendo uma estratégia organizacional de gerenciamento de requisitos em mãos e um campeão no leme, estaremos mais bem preparados para prosseguir com o nosso trabalho. Mas, primeiro, devemos dar uma olhada no *problema de escopo*, o assunto da Habilidade de Equipe 4.

Habilidade de Equipe 4

Gerenciando o Escopo

- **Capítulo 19**: O Problema do Escopo de Projeto
- **Capítulo 20**: Estabelecendo o Escopo de Projeto
- **Capítulo 21**: Gerenciando seu Cliente
- **Capítulo 22**: Modelos de Gerenciamento de Escopo e Processos de Desenvolvimento de Software

Até agora neste volume, introduzimos as Habilidades de Equipe para analisar o problema, entender as necessidades dos usuários, e definir o sistema. Todas as três Habilidades de Equipe tinham o foco nas causas raízes dos problemas de desenvolvimento de software: preparar a equipe para entrar no espaço da solução sem o conhecimento adequado do problema a ser resolvido. Embora os membros da equipe precisem praticar estas habilidades a fim de desenvolvê-las, isso não demandará muito esforço. Recomendamos fortemente que gaste um pouco mais de tempo nestas atividades iniciais; o conjunto total de atividades descritas até aqui deve consumir apenas uma pequena fração do orçamento do projeto, talvez apenas 5% ou mais do custo total. Embora os assuntos sejam complexos, apenas alguns membros da equipe – analistas, gerentes de projeto, líder técnico, campeão de projeto ou gerente de produto – precisam estar fortemente envolvidos neste ponto.

No futuro, no entanto, o jogo muda dramaticamente conforme o tamanho da equipe se eleva significativamente. Cada membro adicionado à equipe deve participar do esforço coordenado da equipe, comunicando-se efetivamente uns com os outros. Além disso, o investimento, ou a taxa de gastos, do projeto se eleva dramaticamente. Criamos documentos dos planos de testes, construímos modelos, refinamos requisitos de projeto, elaboramos use cases, desenvolvemos códigos para que, através disso, possamos criar condições favoráveis de trabalho coordenado e que pode ser mudado se a definição não for bem entendida ou se os requisitos ambientais mudarem.

A pirâmide de requisitos, pela sua forma – larga na base – corretamente sugere que muito mais trabalho nos espera pela frente. A Habilidade de Equipe 4 desenvolve uma estratégia para uma das atividades mais cruciais: gerenciar o escopo. De acordo com dados do Standish Group (1994), *"53% dos projetos irão custar 189% do estimado"*. Os dados de nossa experiência são um pouco pior: quase todos os projetos de softwares se atrasam em 50% a 100%. Assumindo que as outras causas raízes do desenvolvimento de software não serão solucionados do dia para a noite, fica claro que a nossa indústria ou é incompetente ou está tentando fazer muito com muito poucos recursos, habilidades e ferramentas. Nós estamos tentando encher dez pontos de funcionalidades desejadas numa sacola que cabe apenas cinco. Embora a física do desenvolvimento de software não seja clara, é óbvio que este elemento de nossa estratégia é digno de atenção e que a qualidade tanto dos produtos de nosso trabalho quanto de nossa reputação estão em jogo.

Assim, *antes* de aumentar o tamanho da equipe, *antes* de desenvolver especificações mais detalhadas, *antes* de aplicar as idéias tecnológicas ao projeto, e *antes* de construir scripts de testes, devemos parar e aprender como *gerenciar o escopo do projeto*. A psicologia, a tecnologia e o bom gerenciamento de projeto são partes poderosas que esta Habilidade de Equipe irá fornecer para elevar a probabilidade de atingir sucesso em projetos.

Capítulo 19

O Problema do Escopo de Projeto

Pontos chaves

- O escopo de projeto é uma combinação de funcionalidade de produto, recursos de projeto e tempo disponível.
- A lei de Brooks estabelece que adicionar trabalhadores ao projeto de software já em atraso torna-o ainda mais atrasado.
- Se o esforço necessário para implementar as características do sistema for igual a recursos sobre o tempo disponível, o projeto terá um escopo atingível.
- Projetos além do escopo são normais na indústria.
- Em muitos projetos, a fim de fornecer uma razoável probabilidade de sucesso, será necessário reduzir o escopo por um fator de dois.

Como em qualquer atividade profissional, definir compromissos no desenvolvimento de aplicação envolve fazer avaliações realísticas dos recursos de projeto, tempo disponível e objetivos, antes de iniciar as atividades. Para o desenvolvimento de software, esses fatores combinados criam o "escopo" de projeto. O escopo de projeto é uma função:

- da funcionalidade que deve ser liberada para atender as necessidades do usuário
- dos recursos disponíveis do projeto
- do tempo disponível para executar a implementação

A Figura 19–1 fornece uma perspectiva retangular que podemos usar para representar o escopo de projeto.

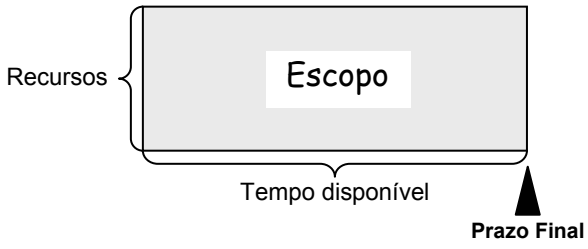


Figura 19–1 Escopo de Projeto

Na Figura 19–1, a área do retângulo representa o escopo atingível do projeto. O Escopo de projeto deriva dos seguintes elementos:

- Os *Recursos* consistem principalmente do trabalho de desenvolvedores, testers, documentadores, pessoal da garantia de qualidade, entre outros.

No início dos anos de 1970, Fred Brooks (1975) demonstrou que adicionar recursos aos projetos de software a fim de aumentar o trabalho realizado é, na melhor das hipóteses, uma proposição de risco. De fato, a lei de Brooks estabelece que adicionar trabalhadores a um projeto de software que está em atrasado torna-o ainda mais atrasado.

Se a escala de tempo for suficiente longa, tudo bem, o trabalho pode realmente ser realizado, mas não será proporcional aos recursos adicionados, e a eficiência geral do projeto tende a diminuir. Adicionar recursos pode até reduzir a velocidade do projeto devido à necessidade de treinar e apoiar as novas pessoas, reduzindo a produtividade daqueles que já estavam no projeto. Como o mercado competitivo nos força a abreviar o tempo disponível, adicionar recursos durante um projeto é impraticável. Além disso, como os orçamentos de desenvolvimento são geralmente apertados nesses anos de Internet, adicionar recursos simplesmente não é uma opção possível.

Com o propósito fazer a análise do escopo, permita-nos assumir que recursos, no eixo y da Figura 19–1, são constantes enquanto durar o projeto.

- O *Tempo* talvez aqui tenha um limite "flexível", sujeito a mudanças se os recursos disponíveis forem inadequados para atingir a funcionalidade desejada. Para o propósito de nossa análise de escopo, o tempo no eixo x, é um fator estabelecido.

  Certamente, a história irá demonstrar que liberar software com atraso é normalmente o "esperado". Por outro lado, muitas equipes de desenvolvimento têm dificuldades de fixar o prazo final. Exemplos disso incluem um programa para declaração de um novo imposto que será liberado definido por lei; apresentação de um novo produto durante uma exposição; ou o prazo final fixado contratualmente pelo cliente. Além disso, se quisermos assegurar nossa credibilidade profissional e ganhar a confidência de nossos clientes, é importante não errarmos no cronograma.

A funcionalidade total que podemos liberar é obviamente limitada pelo tempo disponível (fixado) e pelos recursos disponíveis (também fixados) que temos para usar, assim o escopo disponível é a área do retângulo.

Neste volume nós usamos a noção de "características" para representar o valor funcionalmente adicionado que devemos liberar ao usuário. *Se o esforço necessário para implementar as características requeridas pelo sistema é igual aos recursos sobre o tempo que temos à disposição, o escopo do projeto é atingível, e não teremos problemas*. Salvo circunstâncias inesperadas, a equipe de software irá liberar no tempo sem sacrificar a qualidade.

No entanto, a experiência tem mostrado que existe, com freqüência, uma pobre relação entre esses fatores presentes na equação do escopo. De fato, na turma de requisitos que lecionamos, nós sempre perguntamos: *"No início do projeto, que quantidade de escopo são normalmente disponibilizados pelos seus gerentes, clientes ou stakeholders?"*. Em resposta, apenas alguns iniciantes respondem "inferior a 100%". Os outros apresentam um número que varia de 125% a 500%. A média e a mediana de cada enquête tende à mesma conclusão:

aproximadamente 200%. Este dado tem forte correlação com o que estabeleceu o Standish Group anteriormente, ou seja, que mais da metade de todos os projetos irão custar próximo ao dobro de suas estimativas. Talvez possamos agora entender o por que.

Uma Breve História sobre Escopo de Projeto

Uma vez tivemos uma estudante que tinha sido recentemente promovida para exercer um novo papel, a de gerente de produto para um novo produto de software. Seu currículo incluía muitos aspectos de desenvolvimento de produto e marketing de produto, mas não tinha experiência direta no desenvolvimento de software. Depois de ouvir as respostas para a nossa questão sobre escopo, ela mostrou-se incrédula. Ela olhou ao redor da sala e disse, "Não entendi direito, vocês aprovam projetos com aproximadamente duas vezes a quantidade de trabalho que pode ser razoavelmente realizado num período de tempo disponível? Que tipo de profissão é esta? Vocês estão loucos?" Os desenvolvedores trocaram olhares tímidos e unanimemente responderam, sim.

O que acontece se o projeto continuar com os 200% de escopo inicial?

- Se as características da aplicação forem completamente independentes, não é razoável que apenas a metade deles funcionem quando o prazo final estiver vencido. O projeto está avançando com dificuldades, mas fornece apenas a metade da funcionalidade pretendida. E não é uma metade holística. As características não funcionam em conjunto, e não produzem quaisquer funcionalidades agregadas que sejam úteis. Uma aplicação de escopo drasticamente reduzido é rapidamente remendado e embarcado. Como conseqüência surgem sérias insatisfações por parte dos clientes devido às suas expectativas não terem sido atendidas, compromissos de mercado não poderem ser atendidas, e materiais promocionais e manuais imprecisos terem que ser rapidamente retrabalhados. Toda a equipe fica frustrada e desmotivada.

- No prazo final, apenas 50% de cada característica funciona. Além disso, uma vez que existem certas interdependências dentro de cada característica, no caso mais típico, *nada funciona direito quando o prazo final estiver vencido*. O prazo final está comprometido dramaticamente. Todos os compromissos serão atrasados; um novo prazo final é programado, e uma nova marcha para a morte recomeça. No pior caso, toda a equipe é demitida após realizarem várias horas-extras durante meses a fio; na "fase" final desta primeira tentativa, é declarada a fase chamada "promoção dos que não participaram", e um novo gerente é adicionado ao projeto.

O que acontece com a qualidade do software durante uma destas situações? O código, que foi finalizado às pressas próximo ao final, tem um projeto pobre e possuem erros grosseiros; testes foram reduzidos ao mínimo ou foram totalmente descartados; e as documentações e sistemas de ajuda foram eliminados. Clientes realizam tanto os testes funcionais quanto os de garantia de qualidade. Em pouco tempo, os clientes reagem ao seu extraordinário esforço da seguinte forma:

"Embora tenhamos ficado inicialmente desapontados com o seu atraso (ou com as poucas coisas funcionando comparado às nossas expectativas), agora nós estamos realmente insatisfeitos porque descobrimos que você nos entregou um lixo".

A Difícil Questão

Claramente, para que a equipe de projeto tenha alguma esperança de sucesso, o escopo deve ser gerenciado antes e durante o esforço de desenvolvimento. No entanto, o cenário típico amedronta: *Se começarmos o esforço de desenvolvimento com, de fato, a expectativa de escopo em 200%, será necessário reduzir o escopo do projeto por um fator de 2 se quisermos obter alguma chance de sucesso*.

O dilema da equipe ao enfrentar este problema talvez leve à dura questão enfrentada pela equipe: *O que podemos fazer para reduzir o escopo e manter o cliente satisfeito?* Bem, nem tudo está perdido. Nós iremos cobrir algumas maneiras de lidar com esta questão nos dois próximos capítulos.

Capítulo 20

Estabelecendo o Escopo de Projeto

Pontos chaves

- O primeiro passo para estabelecer o escopo de projeto é definir um baseline de requisitos de alto-nível, um conjunto de características que se pretende liberar numa versão específica do produto.
- O segundo passo é estabelecer o esboço do nível de esforço requerido para cada característica do baseline.
- Depois, estimar o risco para cada características, ou a probabilidade de que ao implementá-las causará um impacto negativo no cronograma e no orçamento.
- Usando esses dados, a equipe estabelece e assegura que o baseline de liberação irá conter as características que são críticas para o sucesso do projeto.

O Baseline de Requisitos

O propósito do gerenciamento de escopo é estabelecer um baseline de requisitos de alto-nível para o projeto. Definimos baseline como

> *o conjunto de características, ou requisitos, que se pretende liberar numa versão específica da aplicação.*

Veja a Figura 20–1. Tanto o cliente quanto a equipe de desenvolvimento devem concordar com o baseline da próxima release. Em outras palavras, o baseline deve:

- Ser ao menos "aceitável" para o cliente.
- Ter uma razoável probabilidade de sucesso, na visão da equipe.

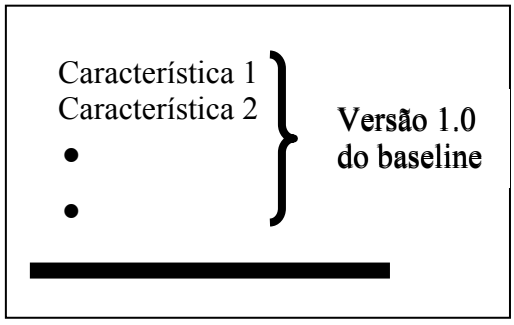


Figura 20–1 Baseline de requisitos

O primeiro passo na criação do baseline é listar as características que terão que ser definidas para a aplicação. Controlar o nível de detalhes neste processo é uma importante chave de sucesso. Na Habilidade de Equipe 3, sugerimos que qualquer novo sistema, não importa o quão complexo seja, pode ser descrito por uma lista de 25 a 99 características. Com mais do que isso, você estará visualizando o projeto num nível de detalhes muito alto, dificultando a comunicação efetiva com clientes e equipe de desenvolvimento. Com menos que isso, o nível de detalhes pode ser muito baixo para fornecer suficiente compreensão da aplicação e do nível necessário de esforço para a implementação.

Se seguirmos o processo de workshop de requisitos (Capítulo 10) ou algum processo que gere resultados similares, teremos à nossa disposição uma lista de 25 a 99 características. Esta lista fornece uma descrição de alto-nível das capacidades de um novo sistema. Esta lista de características é o principal artefato que iremos utilizar para gerenciar o escopo do projeto, antes que investimentos significativos sejam realizados no refinamento de requisitos, projeto, codificação, teste, entre outras atividades.

Por exemplo, considere um produto de software de prateleira com as seguintes características:

Característica 1: Suporte a banco de dados relacional externo

Característica 2: Segurança multiusuário

Característica 3: Habilidade de clonar um projeto

Característica 4: Interface para releases de novos sistemas operacionais (SO)

Característica 5: Assistente de novos projetos

Característica 6: Importação de dados externos por estilo

Característica 7: Implementar ferramenta de dicas

Característica 8: Integrar com o subsistema de gerenciamento de versões

Definindo as Prioridades

Como discutimos em Habilidades de Equipe 2, Entendendo as Necessidades de Usuários, estabelecer as prioridades relativas do conjunto de características é parte integrante do gerenciamento de escopo. Durante a priorização, é importante que clientes e usuários, gerentes de produto, ou outros representantes – exceto a equipe de desenvolvimento – definam as prioridades de acordo com o mercado interno de seus departamentos. De fato, esta priorização inicial deve ser feita sem muita influência da comunidade técnica; caso contrário, o nível de dificuldade para se implementar as características poderá influenciar a priorização realizada pelo cliente, e o resultado do processo poderá ficar comprometido a ponto de não atender as reais necessidades do cliente. Existirá oportunidade para que as questões técnicas sejam consideradas após o processo de priorização. No nosso exemplo de projeto, assumimos que foi realizada uma votação para a priorização das características usando uma escala crítica-importante-útil; os resultados deste exercício estão apresentados na Tabela 20–1.

Tabela 20–1 Características priorizadas

Característica	Prioridade
Característica 1: Suporte a BD relacional externo	Crítica
Característica 4: Interface para releases de novos SO	Crítica
Característica 6: Importação de dados externos por estilo	Crítica
Característica 3: Habilidade de clonar um projeto	Importante
Característica 2: Segurança multiusuário	Importante
Característica 5: Assistente de novos projetos	Importante
Característica 7: Implementar ferramenta de dicas	Útil
Característica 8: Integrar com o subsistema de gerenciamento de versões	Útil

Avaliando o Esforço

A priorização é apenas uma peça do quebra-cabeça do escopo. Afinal de contas, se pudéssemos implementar todas as características de uma só vez, não seria necessária a priorização. No entanto, independentemente de termos ou não o potencial de implementar o conjunto de características de uma só vez, a verdade é que ainda não temos, nem como imaginar, quanto desse total temos reais condições de fazer e, conseqüentemente, não temos condições de traçar o baseline do projeto.

O próximo passo é estabelecer, de maneira aproximada, o esforço necessário para implementar cada uma das características do baseline proposto. Vale lembrar que, nesta fase, existem poucas informações disponíveis; nós ainda não detalhamos os requisitos; muito menos definimos algum projeto no qual pudéssemos utilizar para balizar as nossas estimativas. O melhor que podemos fazer é determinar, "em ordem de magnitude grosseira", o nível do esforço de cada característica.

Estimar esforços nesta fase precoce é uma Habilidade de Equipe que é aprendida. Naturalmente, a equipe de engenharia relutará em fornecer estimativas sem que as características e requisitos tenham sido totalmente entendidos. Porém, este primeiro corte, durante o gerenciamento de escopo, deve ocorrer antes que esses detalhes sejam conhecidos.

Permita-nos assumir que o gerente de produto ou de projeto seja o campeão de nosso projeto e que tenha o seguinte diálogo com os desenvolvedores do projeto[7].

[7] A equipe pode querer usar "equipe-semanas" ou "equipe-meses" como uma estimativa aproximada do impacto total de uma característica sobre o projeto. Esta heurística grosseira serve como substituto para uma estimativa mais detalhada e é comprovadamente melhor do que o resultado deste diálogo. Assim, aos usar estes totais e o total de tempo disponível para o projeto, a equipe pode determinar onde traçar o baseline inicial. Se o baseline contiver o conjunto de características críticas, tudo bem, senão, o projeto estará fora do escopo e um novo projeto, porém menor deverá ser definido.

Gerente de Produto:	*Qual é a dificuldade de implementar esta característica?*
Equipe de Desenvolvimento:	*Nós não sabemos. Nós ainda não temos quaisquer requisitos ou o seu projeto.*
Gerente de Produto:	*Tudo bem; mas é a coisa mais fácil que nós já fizemos?*
Equipe de Desenvolvimento:	*Não.*
Gerente de Produto:	*Certo; é a característica mais difícil da lista?*
Equipe de Desenvolvimento:	*Não.*
Gerente de Produto:	*Numa escala de baixo-médio-difícil, podemos dizer que é médio?*
Equipe de Desenvolvimento:	*Sim; é médio.*
Gerente de Produto:	*Certo; agora vamos avaliar a próxima característica.*

Por que nós não permitimos um processo que crie requisitos e informações detalhadas de projeto para cada característica a fim de permitir a realização de estimativas mais significativas? Isso não seria a maneira profissional de abordar o problema? Se no cronograma houver tempo para permitir realizar estimativas mais detalhadas, então devemos fazê-lo!

No entanto, nós acreditamos que seja crucial estarmos prontos para tomar algumas decisões rápidas sobre o escopo da atividade sem que exista uma estimativa mais detalhada. Por que? Porque caso contrário poderá haver desperdício de recursos investidos na especificação de requisitos e obtenção de informações de projeto que não serão implementados, na confecção de scripts de teste de características que estarão fora do escopo desse projeto, na determinação incorreta do caminho crítico de projeto, entre outros. Não podemos nos permitir investir quaisquer recursos em atividades que produzam sucata, ou falharemos em otimizar os recursos investidos. Em outras palavras, o gerenciamento de requisitos irá reduzir o número de características que será desenvolvido na release inicial, e desde que recursos são extraordinariamente escassos, nós não podemos nos dar ao luxo de investir recursos adicionais em características que não serão implementados no baseline corrente. A Tabela 20–2 ilustra a adição da informação de esforço à nossa lista de características.

Tabela 20–2 Lista de características com adição do esforço

Característica	Prioridade	Esforço
Característica 1: Suporte a BD relacional externo	Crítica	Médio
Característica 4: Interface para releases de novos SO	Crítica	Alto
Característica 6: Importação de dados externos por estilo	Crítica	Baixo
Característica 3: Habilidade de clonar um projeto	Importante	Alto
Característica 2: Segurança multiusuário	Importante	Baixo
Característica 5: Assistente de novos projetos	Importante	Baixo
Característica 7: Implementar ferramenta de dicas	Útil	Baixo
Característica 8: Integrar com o subsistema de gerenciamento de versões	Útil	Alto

Adicionando o Elemento Risco

Um outro aspecto do gerenciamento de escopo é a estimativa do "risco" associado a cada característica. Neste contexto, iremos considerar o risco como sendo a probabilidade de que a implementação de uma característica provocará impacto imprevisível no cronograma e no orçamento. O risco nos dá uma medida relativa do potencial impacto de incluir uma particular característica dentro do baseline de projeto. Uma característica de risco alto tem potencial para gerar impacto negativo no projeto, mesmo se todas as outras características possam ser realizadas dentro do tempo designado.

A equipe de desenvolvimento estabelece o risco com base em alguma heurística conveniente, usando a mesma escala baixo-médio-alto usada para avaliar o esforço. A Tabela 20–3 ilustra a lista de características revisada de nosso exemplo.

Tabela 20–3 Lista de características com adição do risco

Característica	Prioridade	Esforço	Risco
Característica 1: Suporte a BD relacional externo	Crítica	Médio	Baixo
Característica 4: Interface para releases de novos SO	Crítica	Alto	Médio
Característica 6: Importação de dados externos por estilo	Crítica	Baixo	Alto
Característica 3: Habilidade de clonar um projeto	Importante	Alto	Médio
Característica 2: Segurança multiusuário	Importante	Baixo	Alto
Característica 5: Assistente de novos projetos	Importante	Baixo	Baixo
Característica 7: Implementar ferramenta de dicas	Útil	Baixo	Alto
Característica 8: Integrar com o subsistema de gerenciamento de versões	Útil	Alto	Baixo

A estratégia de reduzir riscos varia de projeto a projeto, e não iremos cobrir esse tópico aqui. Para o propósito de gerenciamento de escopo, basta ter consciência sobre os riscos associados com cada característica, tal que decisões inteligentes possam ser tomadas desde o início do projeto. Por exemplo, se uma característica com prioridade *crítica* tiver um *alto* risco, então será obrigatório aplicar alguma estratégia efetiva de redução desse risco. Se a prioridade for *importante* e a característica estiver próxima ao baseline, o item pode ser descartado ou simplesmente desenvolvido "se houver tempo disponível". Não há problemas nesse processo, contanto que nenhum compromisso tenha sido assumido em incluir esse item na release. Se o benefício de um item de *alto* risco for apenas *útil*, considere descartá-lo completamente.

Reduzindo o Escopo

Nós já fizemos substancial progresso. Nós priorizamos um conjunto de características com esforço e risco associados. Note que normalmente não existe correlação entre a prioridade, esforço e risco. Além disso, a vários itens críticos são de baixo esforço; vários itens que são úteis exigem alto esforço. Isso pode ajudar a equipe na priorização de características. Por exemplo, uma característica crítica, com esforço médio e de baixo risco pode ser um candidato para receber recursos de projeto imediato. Entre esses extremos, podemos encontrar o "ponto ideal", onde podemos aplicar nossos recursos pré-definidos de forma a maximizar os benefícios que serão disponibilizados ao cliente. A Tabela 20–4 fornece um pequeno guia para priorizar o desenvolvimento de características críticas com base nesses atributos.

Tabela 20–4 Técnicas de priorização de escopo

Atributos	Considere
Prioridade: Crítica Esforço: Alto Risco: Alto	Alerta! Estabeleça imediatamente uma estratégia de redução de riscos; aloque recursos; concentre-se na sua viabilidade arquitetural.
Prioridade: Crítica Esforço: Alto Risco: Baixo	Concentre-se nos prováveis item cujos recursos são limitados e críticos; aloque recursos imediatamente.
Prioridade: Crítica Esforço: Baixo Risco: Baixo	Considere alocar recursos só por segurança, ou postergue essa alocação para uma próxima oportunidade.

Uma Estimativa Inicial Razoável

Se a equipe utilizar uma estimativa baseada em atividades, mesmo que grosseira, ela poderá determinar o baseline apenas adicionando as estimativas de atividades até que o limite de tempo seja alcançado; assim, a equipe terá estabelecido o baseline do projeto. Normalmente, no entanto, a equipe não tem, sequer, nenhum dado disponível e ainda assim deve fazer um primeiro corte no escopo de projeto. Neste caso, nós ainda não sabemos onde traçar o baseline, mas se a intuição da

equipe for de que o escopo está acima de 100%, a lista terá que ser cortada sem discussão.

O próximo passo é complicado. Se nós assumimos, por exemplo, que as características estão acima de 200% do escopo, o baseline deve ser cortado pela metade ou mais. Como fazer isso?

A primeira consideração é se podemos realizar somente os itens críticos da lista. Pergunte ao gerente de projeto, "Se tudo falhar, podemos garantir ao menos os itens críticos no prazo final?" Afinal de contas, se nós tivermos aplicado corretamente o esquema de priorização, somente mais ou menos um terço dos itens da lista serão críticos para a release. A menos que algumas características críticas representem um nível de esforço desproporcionalmente alto, a resposta deveria ser sim, mesmo se o escopo seja de 200%. Se a resposta é sim, e *em nossa experiência é sempre sim*, até mesmo nesse primeiro corte, podemos iniciar um plano. Se a resposta for não, o projeto está muito acima do escopo (300%-400% ou mais) , e um projeto de menor escopo deve ser definido e repetido o processo de priorização.

Desde que a nossa abordagem foi grosseira, não podemos assegurar quantos item além dos críticos podem ser realizados. As estimativas de esforço, com base nos requisitos mais detalhados e nas estimativas de viabilidade técnica, podem ser usados para refinar o próximo baseline. (Além disso, esta é a hora de fazer o plano detalhado do projeto para validar as suposições que fizemos).

No entanto, pela nossa experiência, traçar o baseline contendo apenas os requisitos críticos e incluir um ou dois itens *importantes*, é suficiente para muitos projetos do mundo real. Deixe que a equipe de desenvolvimento decida quais itens importantes irão incluir durante o progresso do projeto. Clora, isso não é científico. Porém, funciona!

Se as expectativas forem estabelecidas e gerenciadas corretamente, qualquer coisa que possa ser executada num futuro baseline será um bônus. A Tabela 20–5 aplica esta simples heurística ao baseline de nosso projeto exemplo.

Tabela 20–5 Lista de características priorizadas

Característica		Prioridade	Esforço	Risco
Característica 1:	Suporte a BD relacional externo	Crítica	Médio	Baixo
Característica 4:	Interface para releases de novos SO	Crítica	Alto	Médio
Característica 6:	Importação de dados externos por estilo	Crítica	Baixo	Alto
Característica 3:	Habilidade de clonar um projeto	Importante	Médio	Médio
Baseline (as características acima desta linha são características acordadas com o cliente)				
Característica 2:	Segurança multiusuário	Importante	Baixo	Alto
Característica 5:	Assistente de novos projetos	Importante	Baixo	Baixo
Característica 7:	Implementar ferramenta de dicas	Útil	Baixo	Alto
Característica 8:	Integrar com o subsistema de gerenciamento de versões	Útil	Alto	Baixo

As características que estão abaixo do baseline são agora, características futuras e serão consideradas nas próximas releases. Tais características podem, posteriormente, ser promovidas com prioridade maior, levando-se em consideração os itens que poderão ser realizados em médio prazo e nas futuras solicitações do cliente.

Naturalmente, as características não são sempre independentes. Muitas vezes, algumas características abaixo do baseline é parte integrante de algumas características que estão acima do baseline ou podem ser implementadas mais prontamente como um resultado de ter realizado uma outra característica. Ou, quem sabe, a equipe de projeto seja excelente ou esteja com sorte, e realize as características além do cronograma previsto (uma noção concebível!) ou encontre uma biblioteca de classes que torne fácil implementar as características abaixo do baseline. Nesses casos, a equipe deve estar autorizada a trazer tais características para dentro do baseline (incluindo-as na release corrente), refazer a priorização e redefinir o baseline, desde que se sujeitem, naturalmente, a comunicar tais alterações aos clientes.

Dessa forma, a equipe deveria estar apta a criar um plano de projeto, ao menos uma primeira versão aproximação. No entanto, em todos os casos, muitas características desejadas foram cortadas e será necessário gerenciar as expectativas, seja de dentro ou de fora da empresa. Cobriremos este tópico no próximo capítulo. Mas, primeiro, vamos dar uma olhada no caso de estudos e ver se o que a equipe planeja para a release v1.0 do HOLIS.

O Caso de Estudos

Depois de executar o workshop de requisitos, a equipe do HOLIS recebeu a incumbência de avaliar o nível de esforço de cada característica e apresentar um primeiro rascunho do baseline v1.0. Um gerenciamento rigoroso seria aplicado devido às restrições impostas à equipe, incluindo a data limite de ter que disponibilizar um protótipo para uma exposição em Dezembro e a data (rígida) de uma release que deveria estar pronta em Janeiro[8]. A equipe estimou o nível de esforço de cada característica via a heurística alto-médio-baixo e então adicionou a sua avaliação de risco para cada característica. A Tabela 20–6 apresenta a lista total de características, com o acréscimo desses atributos.

Para o próximo passo, a equipe forneceu estimativas grosseiras para cada característica e desenvolveu um plano de projeto detalhado mostrando certas interdependências e marcos de projeto críticos. Além disso, após a negociação com o departamento de marketing, o qual, por sua vez, negociou com Raquel (seu distribuidor internacional), a equipe determinou que, na release 1.0, era adequado internacionalizar apenas a interface de usuário do UCC, o que reduziu imensamente o escopo desta característica. A internacionalização final do software de interface para Programador PC (opcional) poderá esperar até a versão 2.0. Isso fez com que a equipe alterasse a característica 25 de "Interface de usuário internacionalizado" para "Interface UCC internacionalizado" e a adicionasse uma

[8] Durante o desenvolvimento, a equipe decidiu que na prática, tinha até o final de Fevereiro para liberar a versão 1.0 final. Com isso houve a adição de mais 6 semanas cruciais, pois a equipe estava convencida de que precisava realizar modificações finais, com base no feedback obtido na exposição.

nova característica, “Programador PC internacionalizado”, para a lista de características futuras.

Tabela 20–6 Características do HOLIS 2000, com os atributos de esforço e risco

ID	Características	Votos	Esforço	Risco
23	Cenas de iluminação personalizadas	121	Médio	Baixo
16	Configuração do tempo automático de iluminação, etc.	107	Baixo	Baixo
4	Características de segurança pré-definidas, p.ex., lâmpadas e alarmes	105	Baixo	Alto
6	100% de confiabilidade	90	Alto	Alto
8	Fácil de programar, unidade de controle não-PC	88	Alto	Médio
1	Facilidade de programar as estações de controle	77	Médio	Médio
5	Configuração de ausências	77	Baixo	Médio
13	Toda lâmpada pode ser apagada	74	Baixo	Baixo
9	Usar meu próprio PC para programar	73	Alto	Médio
14	Características de entretenimento	66	Baixo	Baixo
20	Portas da garagem fechadas	66	Baixo	Baixo
19	Ligar automaticamente as lâmpadas quando a porta for aberta	55	Baixo	Alto
3	Interface com o sistema de segurança residencial	52	Alto	Alto
2	Facilidade para instalar	50	Médio	Médio
18	Ligar automaticamente as luzes quando alguém bate a porta	50	Médio	Médio
7	Ligar e desligar instantâneo	44	Alto	Alto
11	Poder dirigir cortinas, sombras, bomba d’água e motores	44	Baixo	Baixo
15	Controlar a iluminação via telefone	44	Alto	Alto
10	Interface com o sistema de automação residencial	43	Alto	Alto
22	Modo gradual: aumento ou redução da luminosidade lentamente	34	Médio	Baixo
26	Estação de controle principal	31	Alto	Alto
12	Facilidade de expansão quando remodelado	25	Médio	Médio
25	Interface de usuário internacionalizado	24	Médio	Alto
21	Interface com o sistema de áudio/vídeo	23	Alto	Alto
24	Restauração após falha no fornecimento de energia elétrica	23	N/A	N/A
17	Controle HVAC	22	Alto	Alto
28	Ativação via voz	7	Alto	Alto
27	Apresentação no web site do usuário	4	Médio	Baixo

Então, com base na atividade de estimativa revisada, a equipe propôs o baseline apresentado na Tabela 20–7. Este baseline foi proposto e enviado para todos da equipe executiva, onde Emily, a VP (vice-presidente), tomará a decisão final. Entretanto, antes de fazer isso, ela precisa que a equipe apresente os detalhes do plano de projeto para que ela possa “ver as dependências”. (A equipe suspeita de que, o que ela realmente quer, é ver se a “lição de casa foi realizada” ou se foi apresentado um “saco cheio de lixo” para obter algum folga no cronograma). No final, a decisão foi *sim*, mas o viso da Emily foi, “Nós aceitamos a proposta da release 1.0 do HOLIS, mas vocês devem estar cientes de que o CEO (diretor responsável da empresa), Mark, ordenou ao meu chefe, Jason, o qual me disse: ‘a senhora não poderá falhar na liberação do produto em Janeiro conforme seu comprometimento’”. A Emily, além disso, comentou, “Não estou segura do que ele quis dizer com isso. Acho que ele quis dizer que se nós falharmos, eu estarei comprometida; mas eu não quero descobrir, você quer?”.

Ouvindo claramente as palavras de Emily, os membros da equipe se responsabilizaram em liberar na data e prosseguir com a próxima fase. O próximo milestone no plano de projeto será a iteração de elaboração, a qual incluirá uma prototipação rápida do HOLIS que deverá estar disponível para demonstração até o dia primeiro de Agosto.

Tabela 20–7 Baseline v1.0 do HOLIS 2000

ID	Características	Votos	Esforço	Risco	Comentários de Marketing
23	Cenas de iluminação personalizadas	121	Médio	Baixo	Tão flexível quanto possível
16	Configuração do tempo automático de iluminação, etc.	107	Baixo	Baixo	Tão flexível quanto possível
4	Características de segurança pré-definidas, p.ex., lâmpadas e alarmes	105	Baixo	Alto	Fazer mais pesquisas de mercado
6	100% de confiabilidade	90	Alto	Alto	Atingir tanto quanto possível os 100%
8	Fácil de programar, unidade de controle não-PC	88	Alto	Médio	Fornecer controlador dedicado
1	Facilidade de programar as estações de controle	77	Médio	Médio	Tão fácil quanto possível com o esforço médio
5	Configuração de ausências	77	Baixo	Médio	
13	Toda lâmpada pode ser apagada	74	Baixo	Baixo	
9	Usar meu próprio PC para programar	73	Alto	Médio	Apenas uma configuração suportada em na versão 1.0
25	**Interface UCC internacionalizado**	24	Médio	Médio	Conforme combinamos com o distribuidor europeu
14	~~Características de entretenimento~~	~~66~~	~~Baixo~~	~~Baixo~~	(Não aplicável, incluída na 23)
7	Ligar e desligar instantâneo	44	Alto	Alto	Tornar o investimento inteligente
	Baseline obrigatório v1.0: Todas as coisas acima desta linha devem ser incluídas ou iremos atrasar a release				
20	Portas da garagem fechadas	66	Baixo	Baixo	Talvez tenha pouco impacto no software
2	Facilidade para instalar	50	Médio	Médio	Nível básico de esforço
11	Poder dirigir cortinas, sombras, bomba d'água e motores	44	Baixo	Baixo	Talvez tenha pouco impacto no software
22	Modo gradual: aumento ou redução da luminosidade lentamente	34	Médio	Baixo	Seria bom se tivéssemos
	Opcional da v1.0: Implemente as características acima o quanto você puderem (Cathy)				
	Características Futuras: As características abaixo não fazem parte do desenvolvimento corrente				
3	Interface com o sistema de segurança residencial	52	Alto	Alto	Poderíamos ter ao menos uma interface de hardware? (Eric)
19	Ligar automaticamente as lâmpadas quando a porta for aberta	55	Baixo	Alto	
18	Ligar automaticamente as luzes quando alguém bate a porta	50	Médio	Médio	
15	Controlar a iluminação via telefone	44	Alto	Alto	
10	Interface com o sistema de automação residencial	43	Alto	Alto	
26	Estação de controle principal	31	Alto	Alto	
12	Facilidade de expansão quando remodelado	25	Médio	Médio	
25	Interface de usuário internacionalizado	24	Médio	Alto	
21	Interface com o sistema de áudio/vídeo	23	Alto	Alto	
24	Restauração após falha no fornecimento de energia elétrica	23	N/A	N/A	
17	Controle HVAC	22	Alto	Alto	
28	Ativação via voz	7	Alto	Alto	
27	Apresentação no web site do usuário	4	Médio	Baixo	

Capítulo 21

Gerenciando o Seu Cliente

Pontos chaves

- Gerenciar seu cliente significa engajá-lo no gerenciamento de *seus* requisitos e no *seu* escopo de projeto.
- Clientes que são partes do processo serão responsáveis pelos resultados.
- Fazer o trabalho correto significa fornecer funcionalidade suficiente no tempo certo para atender as reais necessidades do cliente.
- Habilidades de negociação dão apoio indispensável para o desafio de gerenciar escopo.

Engajando Clientes para Gerenciar Seu Escopo de Projeto

A redução do escopo de projeto para adequar ao tempo e recursos disponíveis tem o potencial de criar uma relação antagônica entre a equipe de projeto e seus clientes, cujas necessidades devem ser atendidas. Deixe-me ser honesto. Todos nós já passamos por isto. Felizmente, isso não tem que ser assim. Ao invés disso, *podemos engajar ativamente nossos clientes no gerenciamento de **seus** requisitos e do **seu** escopo de projeto para garantir a qualidade e pontualidade na liberação do software*.

Esta conclusão baseia-se em algumas percepções importantes:

- Os nossos clientes são os principais interessados em atender seus compromissos financeiros com o seu mercado externo. Assim, liberar uma aplicação de alta qualidade e, se necessário, de escopo-reduzido – dentro do cronograma e orçamento – é o maior benefício que a equipe pode fornecer.
- A aplicação, as suas características chaves e as necessidades de negócio pertencem, todos, aos clientes e não à equipe de desenvolvimento da aplicação.
- Precisamos que clientes nos dêem informações para podemos tomar decisões; só eles podem verdadeiramente determinar como gerenciar o escopo e chegar a um resultado útil. Nós somos os seus humildes servos tecnológicos. O projeto é *deles*.

Comunicando os Resultados

Se o escopo do projeto tiver que ser reduzido, assegure-se que o cliente esteja participando desse processo ativamente. Um cliente que faça parte do processo irá se responsabilizar pelos resultados obtidos. Um cliente excluído desse processo

não ficará feliz com os resultados e ira, naturalmente, culpar os desenvolvedores por não terem se empenhado suficientemente.

Engajar o cliente nesse diálogo ajuda a amansar o problema de gerenciar o escopo de forma extremamente gentil na porta de sua casa. E com a filosofia que descrevemos no capítulo anterior, clientes inteligentes irão se comprometer, junto ao seu mercado externo, apenas com os itens críticos incluídos no baseline. As dificuldades advindas do cronograma atrasado e características não implementadas serão evitadas. Quaisquer características extras, implementadas além do baseline, serão recebidas de positivamente, pois estará excedendo as expectativas.

Algumas vezes, a descoberta do problema de gerenciamento de escopo ocorre sem que o cliente esteja engajado; nesse caso, há grande chance de termos que informar o nosso cliente sobre as más notícias. Informar e/ou gerenciar nossos clientes é um processo delicado, pois requer habilidades de negociação e total responsabilidade e comprometimento com o cronograma e escopo apresentados. Após apresentar as más notícias, não podemos nos permitir falhar com a nossa promessa, sob a pena de, no mínimo, perder a nossa credibilidade.

Negociando com o Cliente

Quase todos os processos de negócio requerem negociação. Considere negociar a postergação da data de liberação com um cliente, negociar um valor maior, negociar o aumento anual com o seu gerente, negociar uma cota atingível para a sua equipe de venda, ou negociar recursos adicionais para o seu projeto.

No interesse do seu projeto e dos objetivos de negócio do cliente, você pode precisar negociar o compromisso de escopo para a sua equipe. A equipe deve também, ter em mente que, em muitos casos, que o cliente já pode ter desenvolvido a habilidade de negociação e irá naturalmente usá-la durante as discussões com você e sua equipe. Assim, se você é um líder de equipe, gerente de projeto ou campeão de projeto, você deve *desenvolver muito bem essas habilidades*. A negociação é uma atividade de negócio profissional. Não é um processo particularmente difícil, e pode ser realizado com integridade, graça e estilo. Aproveite a oportunidade para ter algum treinamento nesse processo; seu departamento de recursos humanos provavelmente pode ajudá-lo, ou você pode querer participar de um seminário externo. Se isso falhar, você deve ao menos se familiarizar com algumas regras do jogo. Por exemplo, uma boa revisão do processo de negociação pode ser encontrada em Fisher, Ury, e Patton's (1983) *Getting to Yes*, o qual pode ser lido em poucas horas. Eles recomendam que pequeno guia para a sessão de negociação.

- Comece otimista sem ser irracional.
- Separe as pessoas do problema.
- Concentre-se nos interesses, não nas posições.
- Entenda e mantenha sua posição.
- Invente opções para ganho mútuo.
- Aplique critérios objetivos.

O princípio guia para o gerenciamento de escopo: *não prometa o que provavelmente não possa ser cumprido e liberado*.

Conforme você negocia com o seu cliente, seu princípio guia para estabelecer um baseline deve ser: *não prometa o que provavelmente não possa ser cumprido e*

liberado. Isso assegura que caprichos inevitáveis do desenvolvimento de software, riscos tecnológico não previstos, mudanças de requisitos, atrasos no recebimento de componentes adquiridos, dispensa imprevista de um membro chave da equipe, entre outros, possam ser acomodados dentro do seu cronograma de projeto. Se acontecer de você executar um projeto livre dessas circunstâncias indesejáveis, tudo bem: no pior caso, você terá que se preocupar com a liberação antecipada! Mesmo que seja o de fornecer ao menos alguma festa considerável dentro de sua companhia!

Gerenciando o Baseline

Gerentes de desenvolvimento de bem sucedidos criam margens de erro na estimativa de esforço e levam em consideração o tempo para incorporar mudanças legitimadas durante o ciclo de desenvolvimento. Esses gerentes também resistem às características desagradáveis, as quais o autor Jerry Weinberg (1995) nos lembra que o escopo pode se elevar até 50-100% depois de iniciar o projeto. Ao se concentrar no esforço de desenvolvimento de prioridades críticas do cliente pode aliviar até os ambientes políticos hostis. Com o escopo negociado dentro de um nível atingível, e com o foco de desenvolvimento sempre voltado para o que "deve ter" do cliente, a equipe irá alcançar credibilidade por atender o cronograma com qualidade e, ocasionalmente, com serviços que não podiam ser previstos com antecedência.

Todavia, seus clientes (internos ou externos), querem, naturalmente, a maior quantidade de funcionalidade possível em cada release do sistema de software. Afinal de contas, são as funcionalidades liberadas que adicionam valor necessário para atender a seus objetivos de negócio. De fato, devemos ter um respeito salutar com os clientes que estão gerando a demanda, pois são essas funcionalidades que irão, em última instância, garantir o sucesso no mercado. Clientes competentes, na realidade, são os únicos que merecem ter demanda por funcionalidades.

No entanto, não considerar a demanda por mais e mais funcionalidades pode comprometer a qualidade e a viabilidade de todo o projeto. O *mais* torna-se inimigo do *adequado*. O *melhor* torna-se inimigo do *suficientemente bom*.

Se nós estivermos trabalhando num setor de negócio onde a física esteja mais bem definida, onde a indústria tenha algumas centenas de anos de experiência em liberar produtos, as coisas poderiam ser diferentes. No entanto, nós trabalhamos no mundo do software; onde a física é indeterminada, os processos são imaturos, e a tecnologia é mutável. Permita-nos focar, inicialmente, em como fazer um serviço direito: *funcionalidade suficiente no tempo certo para atender as reais necessidades do cliente*. Nós podemos refinar nosso processo mais tarde para ver se podemos superar as expectativas, mas por agora, vamos apenas nos concentrar em atendê-los! Para isso, precisamos gerenciar o baseline.

Uma vez estabelecido, o baseline fornece o centro focal de muitas atividades de projeto. O baseline de características pode ser usado para avaliar realisticamente o progresso. Recursos podem ser ajustados com base no progresso relativo do baseline. As características dentro do baseline podem ser refinadas em detalhes suficientes para satisfazer o desenvolvimento do código. A rastreabilidade de requisitos pode ser aplicada desde as necessidades do usuário passando pelas

características do baseline. A rastreabilidade pode ser posteriormente estendida das características até as especificações adicionais e implementações.

Talvez o mais importante, o baseline de alto-nível pode ser usado como parte de um processo efetivo de gerenciamento de mudanças. As mudanças são parte integrante de todas as atividades de desenvolvimento. O gerenciamento de mudanças é tão crítico que nós dedicamos para tratar desse assunto no Capítulo 34. Por agora, iremos ver como podemos aplicar o baseline de características para este aspecto importante do gerenciamento de software.

Mudança Oficial

O baseline de características fornece um excelente mecanismo para gerenciar mudanças de alto-nível. Por exemplo, quando o cliente solicita uma nova capacidade do sistema (uma mudança oficial) e essa capacidade não é parte do baseline de características, o impacto dessa mudança deve ser avaliada antes de incluí-la no baseline. Se a equipe de projeto tiver realizado um bom trabalho de definir o baseline antes de começar, a suposição deve ser a de que *qualquer mudança no baseline deve afetar os recursos, o cronograma, ou o conjunto de características* a serem liberadas na release.

Se os recursos são fixos e o cronograma não puder ser alterado, a equipe de projeto deve engajar o cliente no processo de tomada de decisão que priorize a nova característica em relação a outras características programadas para a release. Se a nova característica tiver uma prioridade *crítica*, ela deve, por definição, ser incluída na release, e o cliente e a equipe de projeto devem em conjunto, determinar quais características serão excluídas da release ou ao menos reduzidas em suas prioridades, com a redução associada das expectativas. Se a característica é *importante*, mas não *crítica*, a equipe de projeto pode prosseguir com a implementação da característica na base de grandes esforços, deixando que o progresso dite se a característica fará ou não parte da release.

Mudança Não-oficial

Paradoxalmente, o problema de mudanças iniciadas pelo cliente pode ser o mais fácil de tratar dentre os desafios do gerenciamento de requisitos. O foco do problema é externo, podemos estabelecer certas salvaguardas, e o impacto das mudanças podem ser avaliados e apresentados aos stakeholders externos.

No entanto, a experiência mostra que a ameaça de uma outra classe de mudanças é muito mais subversivo ao processo de desenvolvimento. No Capítulo 34, discutiremos os damos obscuros dessas mudanças e adquirir munição adicional para atacar esse desafio de gerenciamento de escopo.

Capítulo 22

Gerenciamento de Escopo e Modelos de Processo de Desenvolvimento de Software

Pontos chaves

- O processo de desenvolvimento da equipe define *quem* está fazendo *o que*, *quando* e *como*.
- No modelo cascata, as atividades progridem através de uma seqüência de passos, com cada passo apoiado nas atividades do passo anterior.
- O modelo espiral inicia com uma série de protótipos dirigidos pelo risco, seguido por um processo estruturado similar ao modelo cascata.
- A abordagem iterativa, um híbrido dos modelos cascata e espiral, separa as fases do ciclo-de-vida das atividades de software que ocorrem em cada fase.
- Independentemente de qual modelo você use, você deve desenvolver ao menos um protótipo inicial para obter o feedback do cliente.

Até agora nós não falamos muito sobre o processo de desenvolvimento de software como um todo e seu relacionamento com a habilidade da equipe atingir os resultados desejados. No entanto, o gerenciamento de requisitos que seja efetivo não pode ocorrer sem o contexto de um processo de software razoavelmente bem definido que defina o conjunto total de atividades que sua equipe deve executar para liberar o produto de software final. Alguns processos de software são relativamente formais, e outros são informais, mas sempre existe um processo, mesmo que ele não seja rigoroso ou documentado.

O processo de desenvolvimento de software da sua equipe *define quem (quem da equipe) faz o que (que atividade será executada), quando (esta atividade é executada em relação a outras atividades), e como (os detalhes e passos na atividade) normalmente a sua equipe atinge seus objetivos*. O processo de software tem um efeito material na habilidade da equipe desenvolver software no prazo e dentro do orçamento. Neste capítulo, nós veremos alguns dos aspectos de vários tipos de processos de software, ou seja, fases dependentes do tempo e os principais tipos de atividades dessas fases e, então, analisamos como os vários processos afetam os desafios do gerenciamento de escopo.

O Modelo Cascata

Boehm (1988a) disse que, já nos anos 50 (1950), a indústria de software, reconhecendo o custo de descobrir tardiamente defeitos dentro do ciclo-de-vida de software, adotou um modelo lógico e seqüencial, o qual progredia a partir da fase de requisitos, passando pelas fases de projeto, codificação, e assim por

diante. Isso foi o principal avanço obtido sobre os modelos anteriores de duas fases (codificar e corrigir), através do qual programadores escreviam um código inicial e então o corrigia até que não precisasse mais ser corrigido.

Nos anos 70 (1970), Winston Royce (1970), trabalhando na TRW, definiu o que ficou conhecido como o "modelo cascata" de desenvolvimento de software. O modelo cascata melhorou o modelo estritamente seqüencial pelo:

- Reconhecimento da necessidade de loops de feedback entre estágios, onde admitia que o projeto afeta os requisitos, que a codificação do sistema irá fazer com que visitemos novamente o projeto, e assim por diante.
- Desenvolvimento de um protótipo de sistema em paralelo com as atividades de análise e projeto.

Como ilustra a Figura 22–1, no modelo cascata, as atividades de software progridem logicamente através de uma seqüência de passos. O trabalho em cada passo apóia-se em atividades do passo anterior. O projeto segue logicamente os requisitos, a codificação vem depois do projeto, e assim por diante. O modelo cascata foi largamente seguido no passado, por durante duas décadas, e serviu com sucesso como um modelo de processo para projetos de média a larga escala.

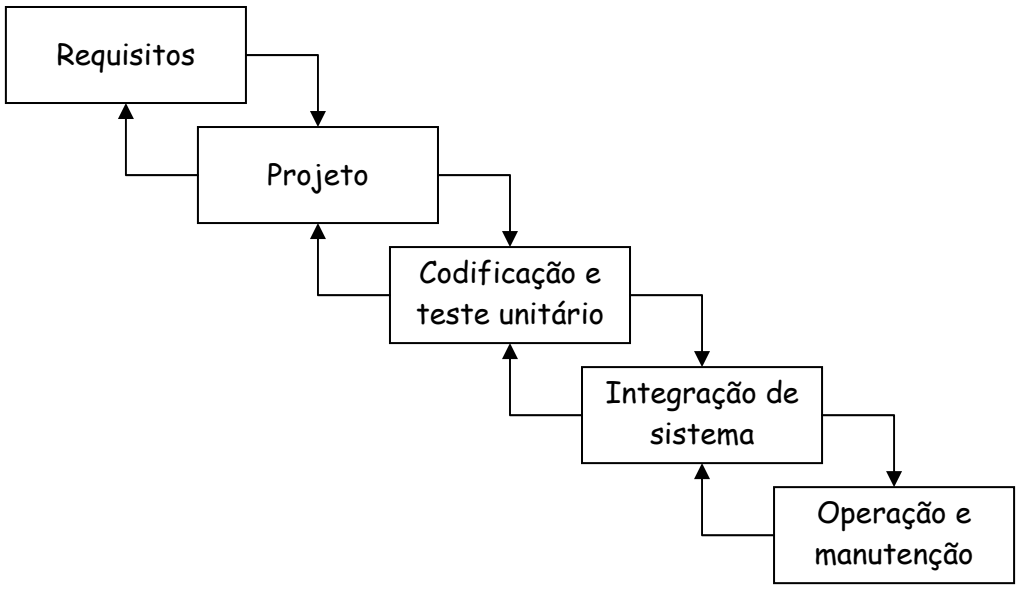


Figura 22–1 Modelo cascata de desenvolvimento de software

Note que, como normalmente ocorre, a Figura 22–1 não referencia a atividades de prototipação como foi prescrito. Isto é um infeliz mistério da história que iremos contar brevemente.

O modelo cascata teve algum sucesso ao reforçar a função dos requisitos como o primeiro passo necessário para o desenvolvimento de software, básico para as atividades de projeto e codificação. Porém, essa força tornou-se também numa fonte de dificuldades, pois deu ênfase na elaboração completa dos requisitos e documentações de projeto, transformando-se numa barreira para sair de cada fase. Além disso, talvez esse grave engano das equipes de desenvolvimento, *tenha feito com que esse modelo representasse uma abordagem fixa e rígida de desenvolvimento, onde os requisitos são "congelados" durante a execução do*

projeto, onde mudanças representam um sacrilégio, e onde o processo de desenvolvimento impõe sua própria vida. Neste caso, com o tempo, a equipe pode ficar completamente desligada do mundo real de onde o projeto originalmente foi concebido.

O modelo cascata encontrou pressão adicional quando foi associado aos desafios do gerenciamento de requisitos (Figura 22–2). Especificamente, se o modelo cascata é aplicado a um projeto que possui inicialmente 200% de escopo, o resultado pode ser desastroso. No prazo final, nada realmente funciona; testes de unidade e de integração do sistema são interrompidos ou abandonados; foram realizados investimentos significativos na especificação, projeto e codificação de características de sistema que nunca serão liberados. Resultado: não existe nada que possa ser liberado, existem apenas o caos, a baixa qualidade e sucata de software.

Devido, principalmente, a essas razões, o modelo cascata tornou-se pouco popular. Um resultado infeliz de tender a saltar direto para o código, mesmo com o entendimento inadequado dos requisitos do sistema, foi um dos principais problemas que estavam tentando resolver no modelo cascata!

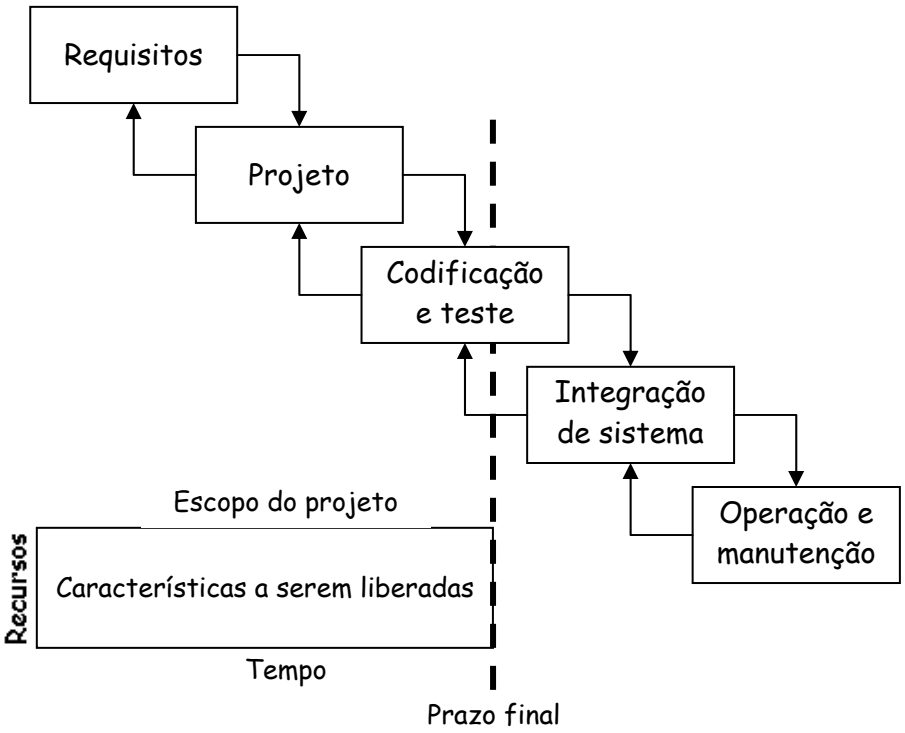


Figura 22–2 Aplicando o modelo cascata num projeto com escopo de 200%

O Modelo Espiral

O principal estudo do Barry Boehm (1988a) recomendava uma estrutura diferente para guiar o processo de desenvolvimento de software. Seu "modelo espiral" de desenvolvimento de software é um modelo funcional para aqueles que acreditam que o sucesso segue um caminho de desenvolvimento mais orientado a riscos e incremental (Figura 22–3).

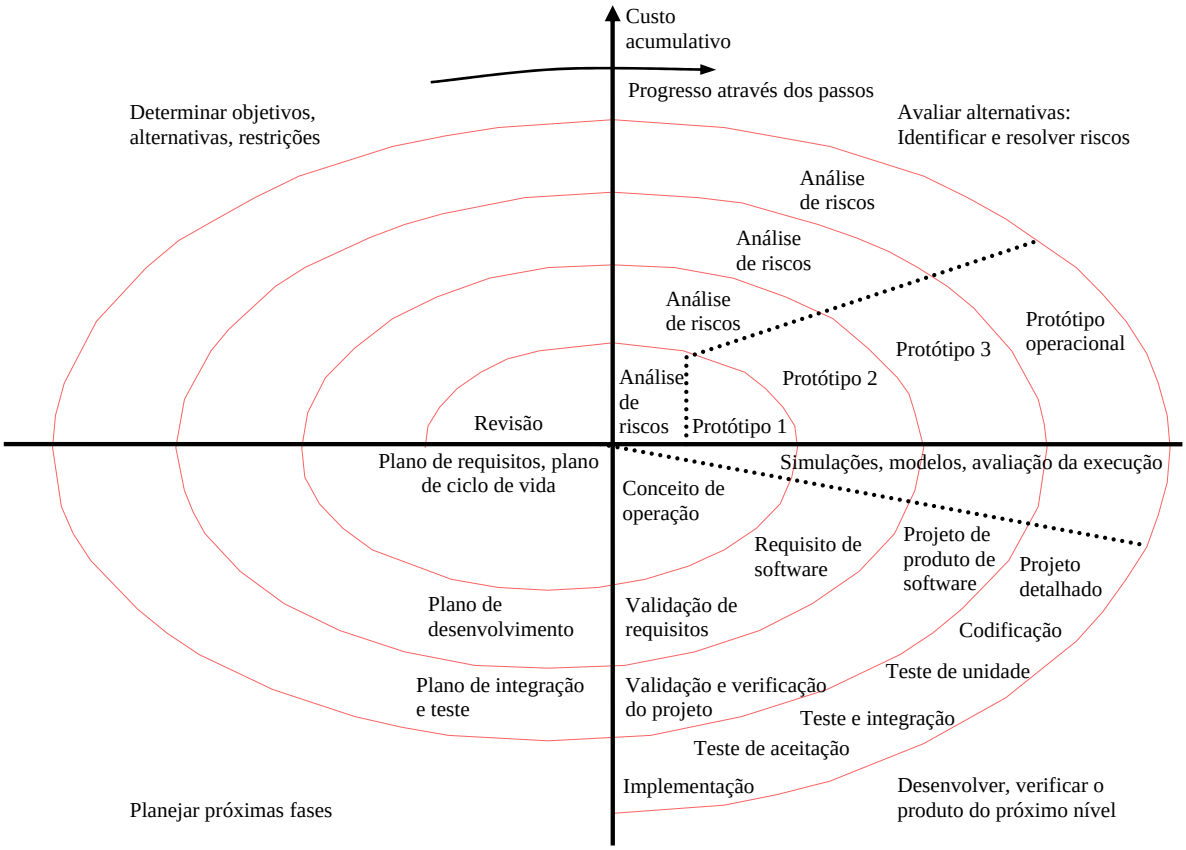


Figura 22–3 O modelo espiral de desenvolvimento

No modelo espiral, o desenvolvimento é inicialmente dirigido por uma série de protótipos orientados a risco; então um processo similar ao modelo cascata é usado para produzir o sistema final. Naturalmente, quando usado inapropriadamente, o modelo espiral pode exibir tantos problemas quanto o modelo cascata usado de forma incorreta. Os projetos algumas vezes caem na abordagem reduzir-e-tentar, liberando incrementos que devem ser expandidos e mantidos com chicletes e barbantes. Alguns a referenciam como o processo de criar instantaneamente códigos legados. O progresso medido pela nossa recente habilidade de criar códigos, incompreensíveis e que não podem ser mantidos, é duas ou três vezes mais rápida do que a tecnologia anterior!

No entanto, quando você olha o modelo espiral com mais cuidado, ele fornece um sensível mapa que ajuda a atacar alguns dos desafios de requisitos apresentados neste volume. Em especial, o modelo espiral inicia com o planejamento de requisitos e validação de conceitos, seguido por um ou mais protótipos para assistir na confirmação precoce de nosso entendimento sobre os requisitos do sistema. A principal vantagem deste processo é a disponibilidade de oportunidades para múltiplos feedbacks de usuários e clientes, dos quais pretendemos obter um "Sim, mas..." logo no início. Oponentes desta rigorosa abordagem dizem que nos ambientes atuais, não podemos nos dar ao luxo de aplicar o tempo para realizar uma validação completa dos conceitos, criar dois ou três protótipos, para então aplicar rigorosamente o modelo cascata.

Retornando à pergunta, o que acontece no modelo espiral quando um projeto é iniciado com um escopo de 200%? A Figura 22–4 ilustra o resultado. Nós podemos dizer que o resultado não é muito melhor do que o plano desastroso quando se utiliza o modelo cascata; outros dizem no prazo final, teremos apenas um ou dois protótipos operacionais e um feedback do cliente. (Naturalmente, esse feedback estará focado na inutilidade dos softwares que estão para ser liberado para produção!).

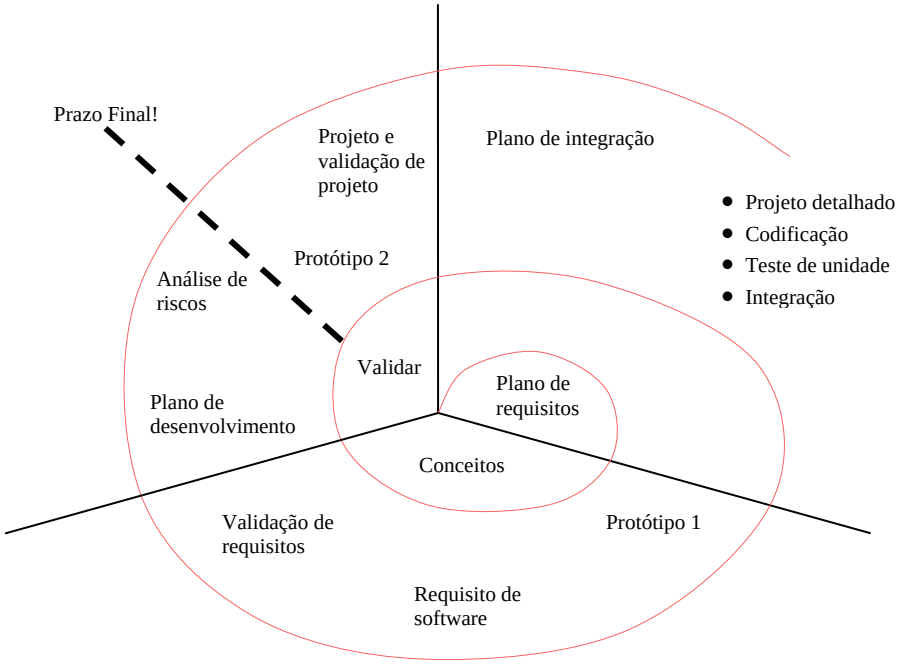


Figura 22–4 Aplicando o modelo espiral num projeto com 200% de escopo

A Abordagem Iterativa

Na década passada, muitas equipes migraram para uma nova abordagem, uma que combina o que existe de melhor nos modelos cascata e espiral e que é um híbrido desses dois. A nova abordagem também incorpora algumas construções adicionais avançadas da disciplina de engenharia de processos de software. A "abordagem iterativa", introduzida por Kruchten (1995), tem sido bem descrito em inúmeros textos, incluindo Kruchten (1999) e Royce (1998). Esta abordagem tem provado a sua efetividade em vários tipos de projetos e pode exibir inúmeras vantagens sobre os modelos de desenvolvimento cascata e espiral.

Nos modelos de desenvolvimento de software tradicionais, o tempo se segue através de uma série de atividades seqüenciais, com o requisito precedendo o projeto, o projeto precedendo a implementação, e assim por diante. Isso parece ser bastante lógico. Na abordagem iterativa, as fases do ciclo-de-vida não estão acopladas às atividades lógicas de software que ocorrem em cada fase, permitindo-nos revisitar as várias atividades, tais como requisitos, projetos e implementação, em várias iterações do projeto. Além disso, como no modelo espiral, cada iteração é projetada para reduzir quaisquer riscos presentes no estágio de atividade de desenvolvimento.

Fases do Ciclo-de-Vida

A abordagem iterativa consiste de quatro fases do ciclo-de-vida: concepção, elaboração, construção e transição, as quais correspondem claramente aos "estados" naturais do projeto ao longo do tempo. (Veja a Figura 22–5).

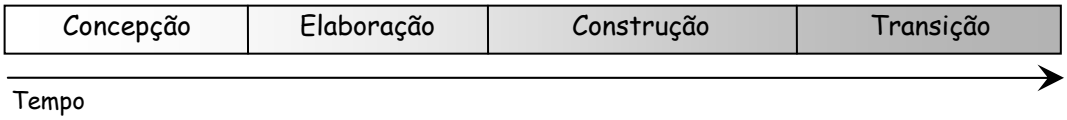


Figura 22–5 Fases do ciclo-de-vida na abordagem iterativa

1. Na *fase de concepção*, a equipe se concentra em entender o caso de negócio do projeto, o escopo do projeto, e na viabilidade de uma implementação. A análise do problema é realizada, o documento da Visão é criado, e estimativas preliminares de cronograma e custos, bem como os fatores de risco de projeto são definidos.
2. Na *fase de elaboração*, os requisitos do sistema são refinados, uma arquitetura inicial executável é estabelecida, e um protótipo de viabilidade inicial é normalmente desenvolvido e demonstrado.
3. Na *fase de construção*, o foco está na implementação. A maior parte do código é desenvolvido aqui nesta fase, e a arquitetura e o projeto é totalmente desenvolvido.
4. O beta teste normalmente ocorre nesta *fase de transição*, e os usuários e pessoal de manutenção do sistema são treinados. O baseline testado da aplicação é implantado para a comunidade de usuários e implantado para uso.

Iterações

Dentro de cada fase, o projeto normalmente experimenta várias iterações (Figura 22–6). Uma iteração é uma seqüência de atividades com um plano e critérios de avaliação estabelecidos, resultando em algum tipo de executável. Cada iteração é construída sobre a funcionalidade da iteração anterior; assim, o projeto é desenvolvido de forma "iterativa e incremental".

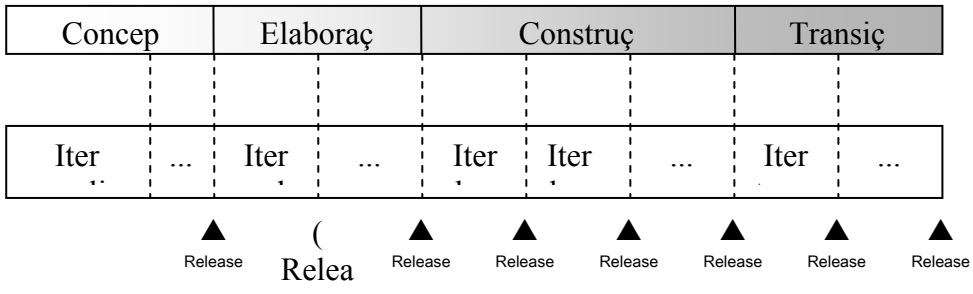


Figura 22–6 Iterações de fases, resultando em releases viáveis

A seleção das iterações depende de vários critérios. As iterações iniciais devem ser projetadas para avaliar a viabilidade da arquitetura escolhida contra alguns dos mais use-cases importantes cercados de riscos.

Workflows

Na abordagem iterativa, as atividades associadas ao desenvolvimento de software são organizadas num conjunto de *workflows*. Cada workflow consiste de um conjunto de atividades logicamente relacionadas, cada um definindo como as atividades devem ser seqüenciadas produzir um produto de trabalho ou "artefato" viável. Embora a quantidade de workflows possa variar, dependendo da empresa ou das circunstâncias do projeto, existem normalmente seis workflows, como ilustra a Figura 22–7.

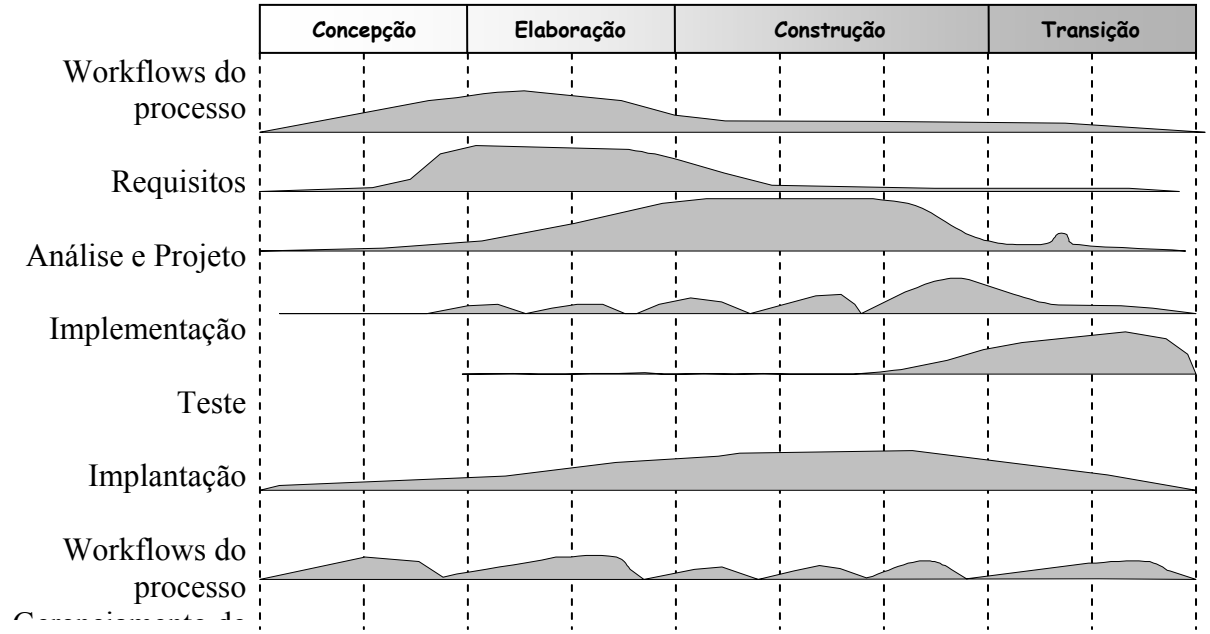


Figura 22–7 Workflows da abordagem iterativa

Durante cada iteração, a equipe gasta tanto tempo quanto apropriado em cada workflow. Assim, uma iteração pode ser considerada como uma mini-cascata através de atividades de requisitos, análise e projeto, e assim por diante, mas cada mini-cascata é "dedicada" às necessidades específicas daquela iteração. O tamanho da "corcova" da Figura 22–7 indica a quantidade relativa de esforço investido num workflow. Por exemplo, na fase de elaboração, tempo significativo é gasto em "refinar" os requisitos e em definir a arquitetura que irá sustentar as funcionalidades. As atividades podem ser seqüenciais (um verdadeiro mini-cascata) ou podem ser executadas concorrentemente, quando apropriado ao projeto.

Da perspectiva de gerenciamento de requisitos, a abordagem iterativa trás duas significativas vantagens:

1. *Obter o "Sim, mas" desde o início*. Cada iteração produz uma release executável tal que, logo no início, clientes têm a oportunidade de ver o produto do trabalho. Claro que a reação do cliente será um "Sim, mas", mas nos estágios iniciais, apenas um mínimo de investimento foi realizado. Conforme as iterações vão sendo realizados, os tamanhos do "Sim, mas" devem decrescer, e você e seu cliente irá, em algum momento, convergir para um sistema correto.

2. *Melhor gerenciamento do escopo*. Se a primeira iteração ultrapassar em 30%, isso é uma indicação de que o escopo do projeto pode estar mal definido, e ajustes podem ser realizados. Mesmo que o escopo não seja bem gerenciado, múltiplas iterações foram desenvolvidas e executadas antes que chegasse o prazo final, e pelo menos foram implantados. Mesmo que falte alguma funcionalidade, a release irá liberar algum valor ao usuário, se as características tiverem sido selecionadas e priorizadas cuidadosamente, permitindo que seu cliente atinja seus objetivos, ao menos em parte, até que você continue com as próximas iterações. E, se a arquitetura for robusta e atender as questões técnicas, sua equipe terá uma plataforma sólida onde funcionalidades adicionais poderão ser construídas.

O que fazer, O que fazer ...

Indep
do m
utilize, a equipe deve fornecer ao menos um protótipo de avaliação robusto a fim de obter feedback do cliente logo no início.

Uma das premissas desse volume é que obter o "Sim, mas" desde o início é uma atividade poderosa e a melhor do processo de software.

- Quantas vezes você tem para fazer isso?
- O cliente solicita mudanças em cada protótipo?
- Estamos condenados independentemente do processo que seguimos?

Resposta: ao menos uma vez, sim, e não. Sim, clientes irão exigir mudanças toda vez que ele ver um protótipo. Não, você não está condenado. A razão é que a *quantidade* de mudanças que ocorrem depois que o cliente tiver a oportunidade de ver, tocar e interagir com a implementação pretendida é pequena comparada à resposta do cliente para o primeiro artefato tangível do processo.

Assim, independentemente do modelo de processo de desenvolvimento utilizado, embora a nossa preferência seja o modelo iterativo em projetos de desenvolvimento, é obrigatório que a atividade de desenvolvimento forneça ao menos um protótipo de avaliação robusto para obter o feedback do cliente antes que a maioria das atividades de projeto e codificação seja executada. (Lembra da atividade de prototipação que Royce (1970) sugeriu em seu primeiro modelo cascata?). Devido à redução da quantidade de mudanças ao nível gerenciável, o desenvolvedor consegue fazer mudanças (normalmente em interfaces de usuário, relatórios e outras saídas) incrementalmente no projeto, garantido uma implementação robusta e de altíssima qualidade. A partir daí, um processo rigoroso de finalizar atividades de projeto, codificação, testes de unidade, e integração de sistema, irá fornecer um sólido fundamento para o produto enquanto que simultaneamente auxilia enormemente nas atividades de garantia de qualidade e teste.

Sumário da Habilidade de Equipe 4

Na Habilidade de Equipe 4, Gerenciando o Escopo, aprendemos que o problema do escopo de projeto é endêmico. Projetos normalmente são iniciados com aproximadamente duas vezes a quantidade de funcionalidades que a equipe pode razoavelmente implementar com qualidade. Isso não deveria nos surpreender, isso faz parte da natureza humana: clientes querem mais, o mercado deseja mais e nós também queremos mais. Nós apenas precisamos garantir que nos impusemos uma dieta suficiente para assegurar que podemos liberar as *coisas* dentro do prazo.

Nós vimos várias técnicas para estabelecer prioridades e definimos a noção de baseline – um acordo sobre o que o sistema irá fazer – um produto chave do projeto – a nossa pedra fundamental e ponto de referência para decisão e avaliação. Aprendemos que, se o escopo e a respectiva expectativa exceder a realidade, em todas as probabilidades, algumas notícias ruins terão que ser dadas. Decidimos adotar uma abordagem filosófica que engaje nosso cliente nas decisões difíceis. Afinal de contas, nós somos apenas os recursos, não os responsáveis por tomar decisões; o projeto é do cliente. Então, a questão é: Dado os recursos que estão disponíveis para o projeto, o que exatamente deve ser realizada na próxima release?

Apesar de tudo, esperamos fazer alguma negociação; a vida e, certamente, o negócio são, num dado sentido, uma negociação e não deveríamos ficar surpresos com isso. Nós mencionamos brevemente algumas habilidades de negociação e demos dicas de quais dessas habilidades a equipe pode precisar nessas ocasiões.

Nós não podemos esperar que este processo elimine os desafios de escopo, muito mais do que qualquer outro processo sozinho irá resolver os problemas do mundo do desenvolvimento de aplicações. No entanto, com os passos que delineamos podemos esperar ter um efeito material sobre o escopo do problema, permitindo que desenvolvedores de aplicação concentrem-se nos subconjuntos críticos e libere incrementalmente sistemas de altíssima qualidade e que atendam ou excedam as expectativas dos usuários. Além disso, ao engajar o cliente para que nos ajude a resolver o problema de gerenciamento de escopo, eleva o comprometimento das partes, estimulando a comunicação e confiança entre o cliente e a equipe de desenvolvimento. Com uma definição compreensiva do produto (documento da Visão) sob controle e o escopo num nível razoavelmente gerenciável, mesmo que no início o escopo esteja exagerado, teremos ao menos a *oportunidade* de ter sucesso nas próximas fases do projeto.

Habilidade de Equipe 5

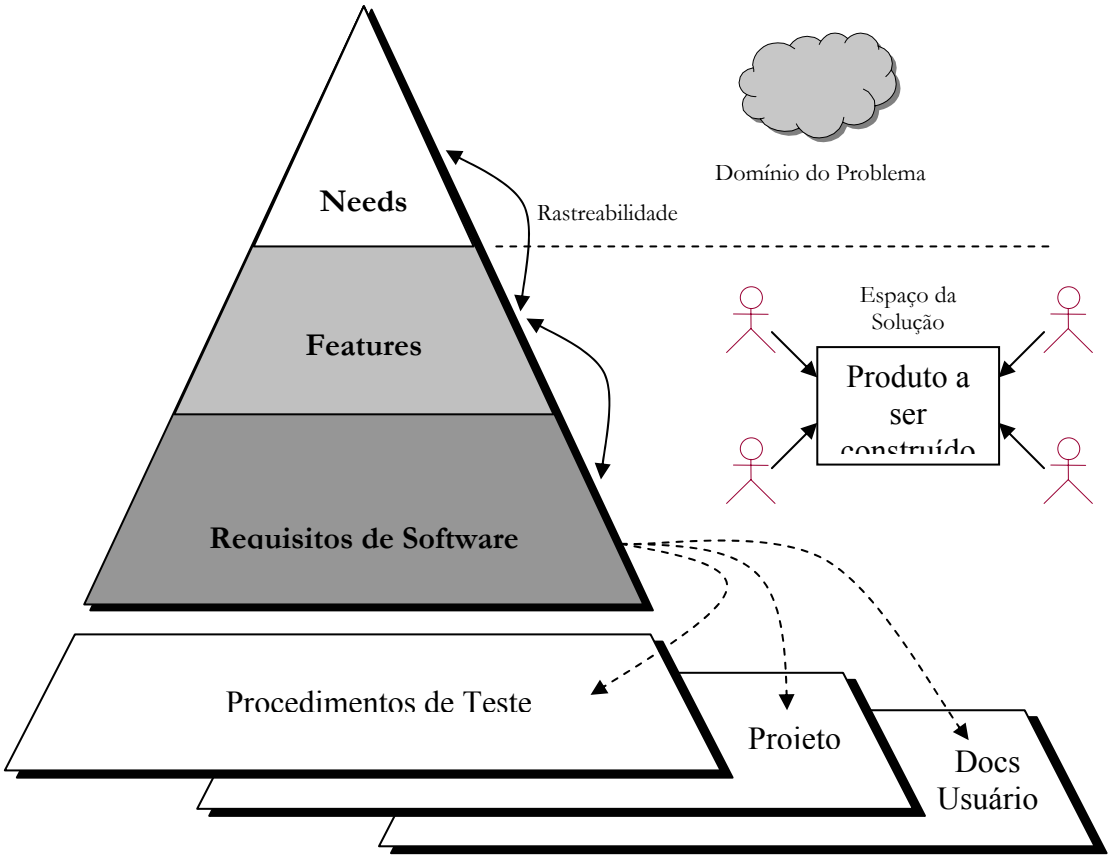
Refinando a Definição do Sistema

- **Capítulo 23**: Requisitos de Software
- **Capítulo 24**: Refinando Use Cases
- **Capítulo 25**: Uma Especificação de Requisitos de Software Moderna
- **Capítulo 26**: Sobre Ambigüidade e Especificidade
- **Capítulo 27**: Medidas de Qualidade de Requisitos de Software
- **Capítulo 28**: Métodos Técnicos para Especificação de Requisitos

Nas Habilidades de Equipe anteriores, nós nos concentramos sobre o processo de analisar o problema, elucidar necessidade do usuário, coletar, documentar e gerenciar as características do produto desejado. Uma vez que as características do produto tenham sido especificadas, a próxima tarefa é refinar a especificação até atingir um nível de detalhes adequado para orientar os processos de projeto, codificação e teste.

Na Habilidade de Equipe 5, examinamos um método organizado para elaborar, organizar e comunicar os requisitos de software. Iremos finalizar a Habilidade de Equipe 5 com uma visão de um assunto mais intrincado: como estabelecer os requisitos de forma clara e concisa.

Independentemente do método que você utilize para coletar os requisitos, é crucial que você adote a filosofia de que os requisitos coletados e somente esses requisitos irão dirigir o projeto. Se descobrirmos que eles são insuficientes ou errados, eles devem ser rapidamente e oficialmente alterados a fim de manter as regras corretas. Desta forma, a equipe toda terá um alvo não-ambíguo e seus esforços podem se concentrar em descobrir e implementar requisitos, minimizando o tempo gasto em coisas sem relevância. Iremos começar examinando a natureza dos próprios requisitos.



Capítulo 23

Requisitos de Software

Pontos chaves

- Um conjunto completo de requisitos pode ser determinado pela definição das entradas, saídas, funções e atributos do sistema, bem como de seu ambiente.
- Os requisitos devem excluir informações relacionadas ao projeto, tais como cronograma, planos de projeto, orçamento e testes, bem como informações de projeto.
- certas opções de projeto, as quais por sua vez, iniciam novos requisitos.
- pelo qual um sistema é desenvolvido.

As características do sistema definidas nas Habilidades de Equipe anteriores, foram deixadas, propositalmente, em alto nível de abstração, para que:

- Pudéssemos entender melhor o aspecto e a forma do sistema através de suas características e de como elas atendem às necessidades do usuário.
- Pudéssemos avaliar o sistema quanto à completeza e consistência e sua adequação em relação ao seu ambiente.
- Pudéssemos usar essas informações para determinar a viabilidade e gerenciar o escopo do sistema antes de fazer investimentos significativos.

Além disso, essa abstração de alto-nível nos permite tomar decisões sobre requisitos excessivamente restritivos logo no início, isto é, as pessoas têm a oportunidade de adicionar suas perspectivas e valor à definição do sistema antes de fecharem a implementação do sistema. Na Habilidade de Equipe 5, Refinando a Definição do Sistema, nossa discussão muda para a elaboração detalhada das características do sistema, o suficiente para assegurar que as atividades de projeto e codificação resultem num sistema que atendem plenamente as necessidades do usuário. Desse modo, nós nos dirigimos para o próximo nível de especificidade e detalhes, e criamos um modelo de requisitos rico e profundo para o sistema que será construído. Naturalmente, criamos também informações mais detalhadas, as quais terão que ser gerenciadas e, para isso, teremos que ser mais organizados para cuidar desses detalhes adicionais.

O nível de especificidade necessária neste próximo passo depende de vários fatores, incluindo o contexto da aplicação e das habilidades da equipe de desenvolvimento. Em sistemas de software para equipamentos médicos de alta segurança, aeronaves, ou comércio online, o nível de especificidade é apropriadamente alto. O processo de refinamento pode incluir mecanismos formais de garantia de qualidade, revisões, inspeções, modelagem, e outros similares. Em sistemas cujas conseqüências sejam menos catastróficas frente a falhas, o nível de esforço será mais modesto. Nestes casos, o trabalho envolvido

está em simplesmente assegurar que a definição está suficientemente precisa quanto compreensiva por todas as partes envolvidas e ainda forneça um ambiente de desenvolvimento eficiente e uma probabilidade "suficientemente alta" de sucesso. O foco está no pragmatismo e na economia, fazendo uma especificação de requisitos que seja apenas o suficiente para garantir que o software desenvolvido é o que o usuário deseja.

Da mesma forma que não existe uma linguagem de programação certa para todas as aplicações, não existe nenhuma maneira correta de desenvolver as especificações mais detalhadas. Diferentes ambientes clamam por técnicas diferentes. Assim, gerentes e engenheiros de requisitos terão que, provavelmente, desenvolver um misto de habilidades para se adaptarem a diversas circunstâncias. Nós aplicamos inúmeras técnicas em várias circunstâncias, desde documentos de requisitos rigorosamente precisos, banco de dados personalizado ou repositório de requisitos até modelos use-case e métodos mais formais. No entanto, o lugar típico do esforço está na especificação em linguagem natural, escrita com clareza suficiente para o entendimento de todos os stakeholders, clientes, usuários, desenvolvedores e testers; mas suficientemente específico ("Os 4 eixos devem ter uma velocidade máxima de 1 metro/segundo") para permitir a verificação e demonstração da conformidade. Antes de começarmos a coletar os requisitos do sistema, nós iremos inicialmente considerar a natureza dos requisitos que você precisará descobrir e definir.

Definição de Requisitos de Software

No Capítulo 2, nós apresentamos esta definição de requisito extremamente simples:

- Uma capacidade de software que o usuário precisa a fim de resolver um problema e atingir seu objetivo.
- Uma capacidade de software que deve ser atendida ou possuída por um sistema ou componentes do sistema para satisfazer um contrato, padrão, especificação, ou outros documentos formalmente impostos.

Requisitos de *software* são aquelas coisas que o software faz em nome do usuário, periférico ou outro sistema. O primeiro lugar a procurar por requisitos de software é ao redor da fronteira do sistema, por coisas que vão "para dentro" e "para fora" do sistema: as interações do sistema com seus usuários.

Para fazer isso, o mais fácil é pensar que o sistema é uma caixa-preta e pensar nas coisas que você gostaria de ter e definir a descrição completa do que a caixa-preta faz.

Além das entradas e saídas, você também precisará pensar em outras características do sistema, tais como desempenho e outros tipos de comportamentos complexos, bem como em outras maneiras de interação do sistema com o seu ambiente (Figura 23–1).

Usando uma abordagem similar, Davis (1999) determinou que nós precisamos de cinco classes principais de coisas para descrever o sistema totalmente:

1. *Entradas do sistema* – não apenas o *conteúdo* de entrada, mas, também, quando necessário, os detalhes dos periféricos de entrada, sua forma, aparência e aspecto, protocolo de entrada. Como a maioria dos desenvolvedores sabe, esta área pode envolver detalhes significativos e pode estar sujeito à volatilidade, especialmente para GUI, multimídia ou ambientes de Internet.

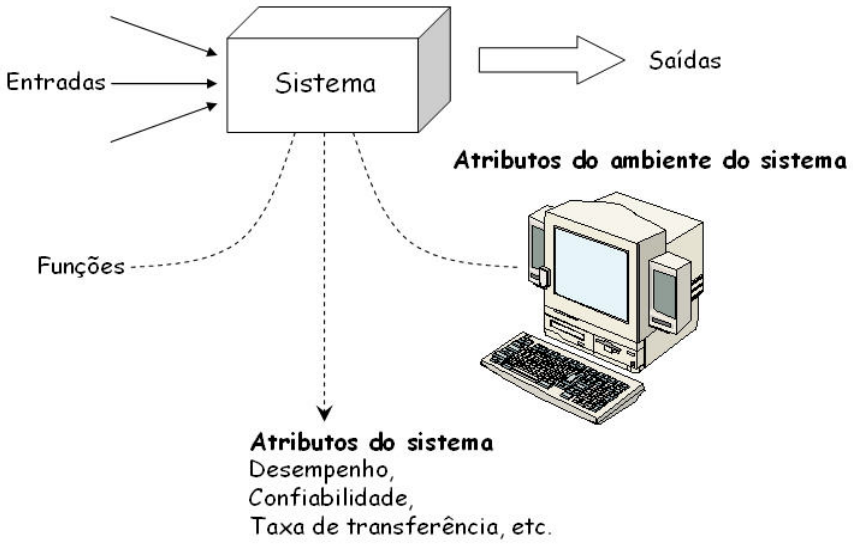


Figura 23–1 Elementos do sistema

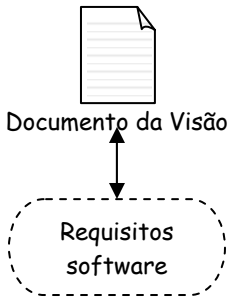
2. *Saídas do sistema* – uma descrição dos periféricos de saída, tais como sintetizador de voz ou apresentação visual, que devem ser suportados, bem como o protocolo e a forma da informação gerada pelo sistema.
3. *Funções do sistema* – o mapeamento das entradas e saídas, e suas várias combinações.
4. *Atributos do sistema* – tais como requisitos não-comportamentais típicos como confiabilidade, facilidade de manutenção, disponibilidade e taxa de transferência, que os desenvolvedores devem levar em consideração.
5. *Atributos do ambiente do sistema* – são os requisitos não-comportamentais adicionais como a habilidade do sistema operar dentro de certas restrições de operação, carga e compatibilidade com o sistema operacional.

Nós usamos esta categorização há vários anos e achamos que funciona muito bem, pois ela nos ajuda a pensar sobre o problema de requisitos de maneira consistente e completa. De acordo com isso, podemos determinar um conjunto completo de requisitos de software definindo:

- Entradas do sistema
- Saídas do sistema
- Funções do sistema
- Atributos do sistema
- Atributos do ambiente do sistema

Além disso, estaremos aptos a avaliar se uma "coisa" é um requisito de software confrontando-a com esta definição.

Relacionamentos entre Características e Requisitos de Software



No início, nós gastamos algum tempo explorando as "características" de um sistema. As características são descrições simples de um comportamento desejado e útil. Agora podemos ver que existe um relacionamento entre características e requisitos de software. O documento da Visão descreve as características na linguagem do usuário. Os requisitos de software, por sua vez, expressam tais características em termos mais detalhados, usando um ou mais requisitos de software específicos descritos pelos desenvolvedores a fim de fornecer as características para o usuário. Em outras palavras, as características nos ajudam a entender e a nos comunicar num nível alto de abstração, mas provavelmente não teremos condições de escrever o código a partir dessas descrições. Elas estão num nível muito alto de abstração para isso.

No entanto, os requisitos de software, são específicos. Podemos codificar a partir dos requisitos de software. Os requisitos de software devem ser suficientemente específicos a ponto de podemos testá-los; isto é, nós devemos estar aptos a testar um sistema verificando que ele realmente implementa os requisitos. Por exemplo, suponha que nós tenhamos desenvolvido um sistema de rastreamento de defeitos para uma empresa de manufatura em linha de montagem ou para uma empresa de desenvolvimento de software. A Tabela 23–1 mostra o relacionamento entre uma das características identificadas no documento da Visão e um conjunto de requisitos associados. Esse mapeamento, e a habilidade de rastreá-los entre as várias características e requisitos, forma a espinha dorsal de um conceito muito importante do gerenciamento de requisitos conhecidos como "rastreabilidade", um tópico que será bem discutido posteriormente.

Tabela 23–1 Requisitos associados a uma característica do documento da Visão

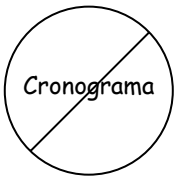
Documento da Visão	Requisitos de Software
Característica 63: O sistema de rastreamento de defeitos irá fornecer informações de tendências para ajudar o usuário a avaliar o status do projeto	RS63.1 A informação de tendência será apresentada num relatório contendo um histograma informando o tempo no eixo-x e o número de defeitos encontrados no eixo-y. RS63.2 O usuário pode entrar com o período da tendência na unidade dia, semana ou mês. RS63.3 Um exemplo do relatório de tendência é apresentado na Figura 1, em anexo.

O Dilema de Requisitos: O que versus Como

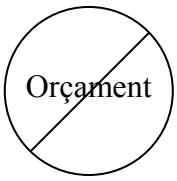
Como podemos ver, os requisitos dizem aos desenvolvedores o que o seu sistema deve fazer e devem envolver assuntos relacionados às entradas, saídas, funções e atributos do sistema, além dos atributos do ambiente do sistema. Mas existe um monte de outras informações que os requisitos *não* deveriam conter. Em particular, devemos evitar estipular quaisquer detalhes de projeto ou de implementação ou informações associadas ao gerenciamento de projeto; e devemos evitar informações sobre como o sistema será testado. Dessa forma, os

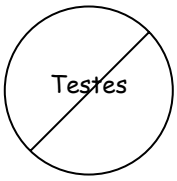
requisitos focam o comportamento do sistema, e sua volatilidade depende da volatilidade do comportamento ou da tendência de mudanças.

Exclua Informações do Projeto



Informações relacionadas ao projeto, tais como cronogramas, plano de gerenciamento de configurações, planos de verificação e validação, orçamento, e horários de turnos, são algumas vezes inseridas no conjunto de requisitos por conveniência do gerente de projetos. Em geral, isso deve ser evitado, uma vez que mudanças nessa informação, tal como mudança do cronograma, eleva a volatilidade e a tendência desses "requisitos" de ficarem desatualizados. (Quando os requisitos são datados, eles se tornam menos confiáveis, com maior probabilidade de serem ignorados). Além disso, debates inevitáveis sobre essas questões deveriam ficar separados da discussão sobre o que o *sistema deve fazer*. Existem diferentes stakeholders envolvidos e cada um serve a propósitos diferentes.





O orçamento pode também ter sido colocado como um requisito; no entanto, é um outro tipo de informação que não satisfaz a nossa definição e, assim, não deve fazer parte dos requisitos gerais do sistema ou do software. O orçamento pode se tornar uma peça importante de informação quando o desenvolvedor precisa decidir qual estratégia de implementação deve escolher, pois algumas estratégias podem ser muito caras ou podem levar muito tempo de execução. Porém, informações de orçamento não são requisitos; da mesma forma, informações que descrevem o como saber que os requisitos foram atendidos – procedimentos de teste ou procedimentos de aceitação – também não atendem a definição e assim, não fazem parte da especificação.

Normalmente, achamos conveniente ir um pouco mais além nesse assunto. Em muitos casos, é útil que os escritores de requisitos dêem dicas de testes disponíveis para requisitos. A final de contas, os escritores de requisitos têm em mente um comportamento específico, e é razoável dar a ajuda que for possível.

Exclua Informações de Projeto

Os requisitos também não devem incluir informações sobre o projeto ou arquitetura do sistema. Caso contrário você pode acidentalmente restringir sua equipe de prosseguir mesmo que as opções de projeto escolhidas façam sentido para a sua aplicação. ("Espere um pouco, temos que projetar isso dessa maneira! Isso não é um requisito!").

Embora a eliminação dos detalhes de gerenciamento do projeto e de testes da lista de requisitos seja, de certa forma, simples, a eliminação de detalhes de projeto/implementação é normalmente mais difícil e muito mais sutil. Suponha, por exemplo, que o primeiro requisito da Tabela 23–1 tenha sido formulada como: "RS63.1 A informação de tendência será apresentada num relatório, escrito em Visual Basic, contendo um histograma informando o tempo no eixo-x e o número de defeitos encontrados no eixo-y" (veja a figura 23–2).

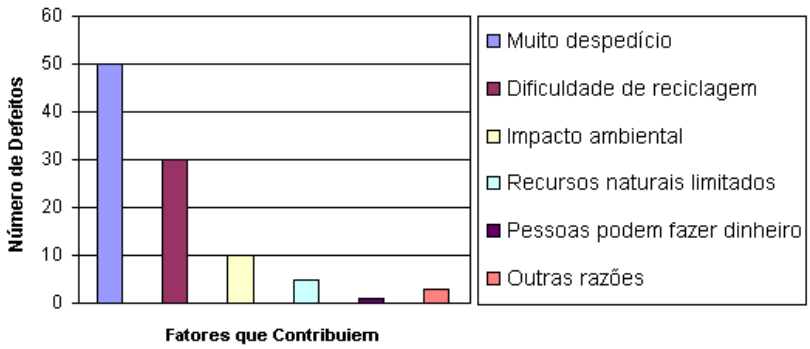


Figura 23–2 Diagrama de Pareto

Embora a referência à linguagem Visual Basic pareça ser uma violação bastante grosseira das orientações recomendadas (pois não representa uma entrada, saída, função, ou atributo comportamental), é útil perguntar: "Quem decidiu impor o requisito de que o histograma deve ser implementado em Visual Basic, e por que foi tomada essa decisão?". As possíveis respostas para esta questão podem ser:

- Um dos membros do grupo, com alguma orientação técnica e que está definindo o documento da Visão, decidiu que o Visual Basic deveria ser especificado porque é a "melhor" solução para o problema.
- O usuário pode ter especificado. Sabendo que a tecnologia pode gerar prejuízos, o usuário quis que a tecnologia VB fosse usada. O usuário não quer que uma outra tecnologia, uma mais cara ou menos disponível, fosse adotada pelas pessoas técnicas, pois ele sabe que o VB está disponível e é relativamente barato.
- Uma decisão política, interna da organização de desenvolvimento, pode ter forçado o uso do Visual Basic para todas as aplicações que serão desenvolvidas. Para assegurar a conformidade e prevenir que esta política não seja ignorada, gerentes insistem em colocar Visual Basic sempre que possível nos documentos de requisitos.

Se um desenvolvedor técnico decidiu inserir uma referência ao Visual Basic devido a uma preferência arbitrária por esta linguagem, é óbvio que essa referência não ocupa um local legítimo na lista de requisitos. Se o usuário forneceu o requisito, a coisa começa a ficar um pouco mais difícil. Se o cliente se recusar a pagar pelo sistema amenos que seja escrito em Visual Basic, o melhor a fazer é tratá-lo como um requisito, *mas um requisito de uma classe especial chamada restrições de projeto, de tal forma a ficar separado dos requisitos normais*. No entanto, é uma restrição de implementação que foi imposta à equipe de desenvolvimento. (A propósito, se você achar que esse exemplo não é realístico, considere um requisito comum que o Departamento de Defesa Norte-Americana impunha em seus contratos de software até o final dos anos 90, onde se exigia o uso da linguagem Ada na construção de sistemas).

A discussão sobre a referência do Visual Basic neste exemplo pode ter obscurecido uma análise de requisitos mais sutil e talvez a mais importante: Por que a informação de tendência tem que ser exibida num relatório com um histograma? Por que não um gráfico de barras, gráfico de setores, ou uma outra forma de representação? Além do mais, a palavra relatório significa um documento impresso em papel, ou também significa que a informação pode ser exibida na tela do computador? É preciso capturar a informação de forma que ela

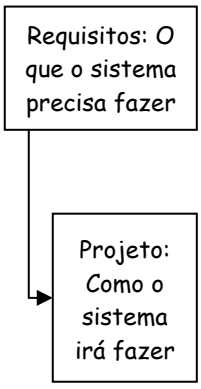
possa ser importada por outros programas ou enviadas para uma extranet corporativa? As características descritas no documento da Visão podem ser sempre preenchidas de várias maneiras, alguma das quais trazem várias conseqüências à implementação.

Em muitos casos, a descrição de um problema, a partir do qual um requisito pode ser formulado, é influenciada pela percepção do usuário da potencial solução disponível para resolver o problema. O mesmo é verdade para os desenvolvedores que participam com o usuário na formulação das características que compõem o documento da Visão e de requisitos. Como um antigo provérbio nos lembra, "se sua única ferramenta é um martelo, todos os seus problemas irão se parecer com um prego". Mas precisamos ficar atentos nas restrições de implementação desnecessárias e inconsistentes infiltradas nos requisitos; sempre que pudermos, precisamos removê-las.

Mais sobre Requisitos versus Projeto

Até agora, nós falamos dos requisitos de software, decisões de projeto e restrições de projeto como se eles fossem entidades distintas que podem ser claramente diferenciados. Isto é, nós estabelecemos ou deixamos implícito que:

- Requisitos precedem o projeto
- Usuários e clientes, por estarem próximos das necessidades, tomam as decisões sobre os requisitos.
- Os técnicos tomam as decisões de projeto porque eles estão melhores preparados para escolher, entre as várias opções de projeto, qual opção atende melhor às necessidades.



Este é um bom modelo, e é o ponto de partida correto para uma filosofia de gerenciamento de requisitos. Davis (1993) chamou isso de o paradigma "o que versus como", onde o "o que" representa os requisitos, ou o "o que" o sistema deverá fazer, e o "como" representa o projeto que será implementado para atingir este objetivo.

Nós contamos essa estória dessa maneira por uma razão. É melhor entender os requisitos antes de projetá-los, e muitas restrições ("use a biblioteca de classes XYZ para acessar o banco de dados") são decisões de projeto importantes registrados nas propriedades dos requisitos, de tal forma que podemos assegurar que os alcançamos as razões contratuais, ou talvez uma mais legítima, as razões técnicas.

Se não pudermos fazer essa diferenciação de alguma forma, o quadro deve ficar muito confuso, e não poderemos diferenciar os requisitos das informações de projeto. Além disso, não saberíamos *quem* é o responsável por *o que* no processo de desenvolvimento. No pior caso, nossos clientes irão ditar o projeto, e nossos projetistas irão ditar os requisitos.

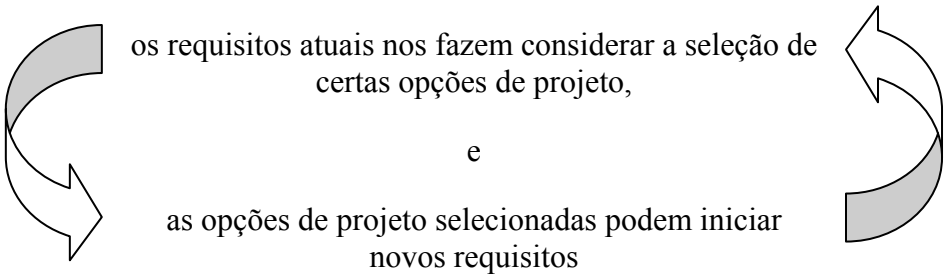
Mas uma complicação sutil e ainda mais séria fundamenta esta discussão e camufla esse simples paradigma que apresentamos. Retornando ao nosso exemplo de estudo de casos, se a equipe toma uma decisão de projeto, tal como a seleção de uma tecnologia PC para executar o subsistema UCC do HOLIS, provavelmente traria algum impacto externo ao usuário. Por exemplo, uma tela de

aviso ou de logon seria exibida em algum momento no mundo do usuário. Melhor ainda, provavelmente nós iríamos querer tomar vantagens de algumas capacidades de entrada de dados do SO, e essas bibliotecas de classe certamente iria exibir seu comportamento externos ao usuário: (Nota para o técnico dentro de você: Sim, nós podemos escondê-lo, mas isto está fora de questão).

Dada as definições fornecidas neste capítulo, a questão é: Uma vez que o impacto de uma decisão de projeto causa comportamento externo visível ao usuário, esta mesma decisão, a qual afeta claramente "entradas e saídas do sistema", agora se torna um requisito? Podemos argumentar que a resposta correta é "Sim" ou "Não", ou até "isso não é realmente importante", dependendo da interpretação individual das definições e análises que fizemos até agora. Mas isso acende um assunto muito importante, dependendo do entendimento sobre o assunto, torna-se crítico entender a natureza do próprio processo iterativo. Vamos fazer um exame mais minucioso.

Iteração de Requisitos e Projeto

Na realidade, as atividades de requisitos versus projeto devem ser iterativos. A *descoberta* de requisitos, sua definição e *decisões de projeto* são circulares. O processo é um contínuo vai-e-vem, onde:


os requisitos atuais nos fazem considerar a seleção de certas opções de projeto,
e
as opções de projeto selecionadas podem iniciar novos requisitos

Ocasionalmente, a descoberta de uma nova tecnologia pode nos fazer abandonar várias suposições sobre o que fazem os requisitos; nós podemos ter descoberto uma abordagem totalmente nova que dispensa a estratégia antiga. ("Vamos abandonar todo o módulo cliente / acesso a dados / GUI e substituí-lo por uma interface baseada em browser"). Essa é uma excelente fonte, e legítima, de mudança de requisitos.

Este processo é como deveria ser; tentar fazer de outra forma seria uma estupidez. Por outro lado, existe um sério perigo nisso tudo, pois se não entendermos verdadeiramente as necessidades do cliente *e* o cliente não estiver ativamente engajado no processo de requisitos – e por que não, em alguns casos, até entender as *nossas atividades relacionadas ao projeto* – decisões *erradas* poderão ser tomadas. Quando gerenciado apropriadamente, esta "reconsideração contínua de requisitos e projeto" é um processo verdadeiramente fantástico, enquanto a tecnologia dirigir a nossa habilidade de, continuamente, melhor atender as reais necessidades de nossos clientes. *Essa é a essência do que é o gerenciamento efetivo e iterativo*. Mas quando gerenciado inapropriadamente, ficaremos continuamente "seguindo a cauda da tecnologia", resultando num desastre. Nós nunca falamos que seria fácil.

Uma Caracterização Avançada de Requisitos

A discussão precedente sobre requisitos sugeriu que existiam vários "tipos" de requisitos. Especialmente, nós achamos que seja útil pensar em três "tipos" de requisitos, como ilustra a Figura 23–3:

- Requisitos funcionais de software
- Requisitos não-funcionais de software
- Restrições de Projeto

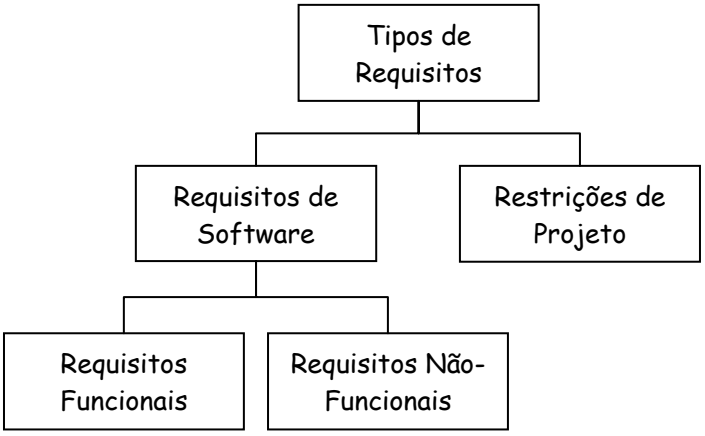


Figura 23–3 Tipos de Requisitos

Requisitos Funcionais de Software

Como seria de se esperar, requisitos funcionais expressam o como se comporta o sistema. Esses requisitos são normalmente orientados a ação ("Quando o usuário faz x, o sistema fará y"). Muitos produtos e aplicações, concebidas para fazer algo útil, são ricos em requisitos funcionais de software. O software é usado para implementar as principais funcionalidades.

Quando você estiver definindo requisitos funcionais, você deve encontrar um bom equilíbrio entre o quão especifico e o quão genérico ou ambíguo você quer que seja esse requisito. Por exemplo, normalmente não é útil ter um requisito funcional declarado na forma: "Quando você pressionar o botão, o sistema é ativado e inicia a operação". Por outro lado, uma declaração de requisito que ocupe várias páginas de texto é, provavelmente, muito específico, mas pode estar correto em vários casos especiais. Nós iremos retornar a essa discussão no Capítulo 26.

Nós achamos que a maioria dos requisitos funcionais pode ser descrita de forma declarativa ou na forma de use cases, a qual iremos descrever no próximo capítulo. A experiência nos tem mostrado que uma ou duas sentenças declarativas de um requisito é normalmente a melhor maneira de satisfazer as necessidades do usuário com um nível de especificidade que o desenvolvedor pode lidar.

Requisitos Não-Funcionais de Software

Até aqui neste capítulo, a maioria dos exemplos de requisitos envolvia requisitos *comportamentais*, ou *funcionais*, de um sistema, concentrando-se nas entradas, saídas, e detalhes de processamento. Os requisitos funcionais nos dizem como o sistema deve se comportar quando apresentado a certas entradas ou condições.

Mas isso não é suficiente para descrever todos os requisitos de um sistema. Nós devemos considerar, também, coisas que Grady (1992) chamou de "*requisitos não-funcionais*":

- Usabilidade
- Confiabilidade
- Desempenho
- Suportabilidade

Esses requisitos são usados com maior freqüência para expressar alguns "atributos do sistema" ou "atributos do ambiente do sistema", elementos de nossa definição elaborada. Esta classificação conveniente nos ajuda a entender mais sobre o sistema que estamos construindo. Vamos ver cada um desses atributos em detalhes.

Usabilidade: É importante descrever o quão fácil o sistema pode ser aprendido e operado pelos pretensos usuários. Além disso, podemos precisar identificar as várias categorias de usuários: iniciantes, usuários normais, usuários experientes, usuários analfabetos, usuários que não são fluentes na linguagem nativa dos usuários normais, entre outros. Se você quer que o seu cliente revise e participe desta discussão – o que seria muito bom – você tem que estar ciente de que, independentemente do requisito que você esteja escrevendo nesta área, terá que ser escrito em linguagem natural; você não deveria querer uma máquina de estado finito para descrever a usabilidade!

Desde que a usabilidade tende a estar nos olhos de quem a vê, como faremos para especificar esse conjunto de requisitos tão confuso? Seguem algumas sugestões.

- Especifique o tempo requerido de treinamento para o usuário se tornar ligeiramente produtivo – apto a realizar tarefas simples – e operacionalmente produtivo – apto para realizar tarefas normais do dia-a-dia. Como pode se notar, podemos precisar descrever posteriormente em termos de usuários novatos, aquele que nunca viu um computador antes, até usuários normais e usuários especialistas.

- Especificar o tempo de tarefas